Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Viki Petrovič

Razvoj odzivne dinamične spletne aplikacije z uporabo AngularJS in Java EE na primeru družbenega omrežja

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: prof. dr. Matjaž B. Jurič

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Preglejte področje razvoja odzivnih dinamičnih spletnih aplikacij. Analizirajte tehnologije na strani strežnika in odjemalca. Proučite HTML5, CSS in Bootstrap. Opišite koncept enostranskih aplikacij in podrobno obdelajte vzorec MVC. Opišite AngularJS in Java EE. Z uporabo omenjenih pristopov in tehnologij razvijte odzivno dinamično spletno aplikacijo na primeru družbenega omrežja.

Izjava o avtorstvu diplomskega dela

Spodaj podpisana Viki Petrovič, z vpisno številko **63100340**, sem avtor diplomskega dela z naslovom:

Razvoj odzivne dinamične spletne aplikacije z uporabo AngularJS in Java EE na primeru družbenega omrežja

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelala samostojno pod mentorstvom prof. dr. Matjaža B. Juriča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 15. septembra 2014 Podpis avtorja:

Zahvaljujem se mentorju prof. dr. Matjažu B. Juriču in njegovim sodelavcem za pomoč in strokovnost pri nastajanju tega diplomskega dela ter družini za podporo, ki so mi jo nudili tekom študija.

Kazalo

Seznam uporabljenih kratic

kratica	angleško
HTTP	Hypertext Transfer Protocol
HTML	HyperText Markup Language
TCP	Transmission Control Protocol
CSS	Cascading Style Sheets
AJAX	Asynchronous JavaScript and XML
FTP	File Transfer Protocol
SSL	Secure Sockets Layer
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language
API	Application Programming Interface
URI	Uniform Resource Identifier
JSON	JavaScript Object Notation
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
JPA	Java Persistence API
EJB	Enterprise JavaBeans
JMS	Java Message Service
JSF	JavaServer Faces
CDI	Contexts and Dependency Injection
JSP	JavaServer Pages
JPQL	Java Persistence Query Language
SQL	Structured Query Language

Povzetek

Namen diplomskega dela je predstaviti razvoj spletnih aplikacij z uporabo sodobnih arhitekturnih pristopov. Zaradi izjemne širine je to področje redko predstavljeno kot zaključena celota, kar je ravno to, kar poskuša doseči to diplomsko delo. Pri tem se delo natančneje osredotoča predvsem na razvoj aplikacij, kjer je večja mera funkcionalnosti implementirana na odjemalcu. V delu je predstavljen razvoj, ki sledi principu model-pogled-kontroler in teži k čim šibkejši sklopljenosti sistema. Opisane so nekatere izmed tehnologij, s katerimi lahko implementiramo omenjene principe, razloženo je, kako se tehnologije med seboj povezujejo in kaj so njihove prednosti in slabosti. Kot pomemben koncept razvoja sodobnih spletnih aplikacij to diplomsko delo predstavi tudi področje odzivnega spletnega oblikovanja. Vse zbrane informacije in ugotovitve tega diplomskega dela so na koncu združene in preizkušene v praksi na primeru gradnje družbenega omrežja.

Ključne besede: spletna aplikacija, enostranska aplikacija, HTML5, MVC, AngularJS, Java EE.

Abstract

The purpose of this thesis is to present the development of web applications by using modern architecture principles and technologies. The main focus of this work is the development of client-centered applications, which gives a good insight into a segment of web development that is currently growing fast. This thesis explains the model-view-controller principle and describes technologies, used to implement it. It also shows how all these technologies fit together. This thesis also explains the concept of responsive web design, which represents an important part of modern web applications. At the end we used the results and identified good practices and patterns to implement a dynamic client-centered social network web application.

Keywords: web application, single-page application, HTML5, MVC, AngularJS, Java EE.

Poglavje 1

Uvod

Z razvojem tehnologije je svetovni splet postal središče dogajanja in pojavljati so se začele nove oblike druženja in povezovanja. V zadnjem času se je opazno razmahnila uporaba družbenih omrežij, ki niso več le prostor za druženje in vzdrževanje stikov, ampak tudi način za oglaševanje storitev in znanj ali za iskanje le-teh. S pojavom družbenih omrežij se je torej začelo odpirati veliko novih možnosti za neposredno povezovanje ljudi in medsebojno sodelovanje. Prav zaradi tega so se nedavno začeli razvijati tudi povsem novi načini iskanja storitev in znanj, znanih tudi pod tujim imenom crowdsourcing. Ta model obhaja klasične ponudnike storitev in znanj, saj poskuša združevati znanje in sposobnosti množice z namenom reševanja določenega problema. Tako lahko z uporabo interneta in množice uporabnikov rešujemo mnogo problemov, tako vsakdanjih kot tudi kompleksnejših oziroma bolj strokovnih.

Poleg družbene revolucije je splet prinesel tudi tehnološko revolucijo. Čeprav je internet prisoten že dalj časa, so spletne aplikacije in omrežja šele v zadnjih letih dosegla pravo razsežnost. Število uporabnikov spleta je močno naraslo, računalniki so postali zmogljivejši in s tem tudi programska oprema, povečala pa se je tudi hitrost interneta. Spletne aplikacije so tako postale pomemben del našega vsakdanjika, zahteve in želje uporabnikov pa postajajo vedno bolj kompleksne. S tem naraščata tudi kompleksnost in obsežnost so-

dobnih aplikacij, ki so hkrati omejene s tehnološkimi značilnostmi, po drugi strani pa morajo izpolnjevati uporabniške zahteve. Zato se je pojavilo veliko orodij, ki razvoj takih aplikacij olajšajo in omogočajo lažjo ali hitrejšo implementacijo naprednejših funkcionalnosti.

Poleg razvoja spleta so razmah doživele tudi pametne mobilne naprave, kot so telefoni in tablice. Vedno več dostopov do spleta je opravljenih iz pametnih naprav, to število pa se bo po napovedih še povečalo. S tem postaja vedno bolj pomembno tudi vprašanje dostopnosti spletnih strani in aplikacij vsem uporabnikom spleta, ne glede na napravo, ki jo uporabljajo.

Cilj tega diplomskega dela je zato raziskati in ovrednotiti sodobne spletne tehnologije, ki omogočajo izdelavo dinamičnih spletnih aplikacij, ki uporabniku nudijo kar najboljšo uporabniško izkušnjo. Pri tem se bo to diplomsko delo osredotočilo predvsem na razvoj spletnih aplikacij, kjer je večina funkcionalnosti implementirana na odjemalcu, kar postaja vedno večji trend v svetu spletnega programiranja.

Poleg vrednotenja tehnologij in predstavitve arhitekturnega pristopa, katerega rezultat ni več lahek, temveč debel odjemalec, je cilj tega dela tak pristop k razvoju tudi preizkusiti v praksi. Izbrane tehnologije bodo uporabljene pri implementaciji družbenega omrežja, ki ga uporabniki lahko uporabljajo za nemonetarno izmenjavo znanj in storitev. S tem torej želi to diplomsko delo ponuditi podrobnejši vpogled v del sicer izjemno širokega področja razvoja sodobnih aplikacij.

V diplomskem delu je najprej na splošno predstavljeno področje spletnega razvoja tako na strani odjemalca kot tudi na strani strežnika. Temu poglavju sledi predstavitev sodobnih spletnih tehnologij, ki so uporabljene tudi pri praktični implementaciji družbenega omrežja. Teoretičnemu pregledu področja sledi prikaz razvoja spletne aplikacije na praktičnem primeru, ki je opisan v zadnjem poglavju.

Poglavje 2

Razvoj spletnih aplikacij

Z nastankom interneta so se pojavile nove možnosti za povezovanje in izmenjavo informacij, tekom razvoja interneta pa se je začelo razvijati tudi področje spletnega programiranja oziroma razvoja spletnih aplikacij. Če so na začetku obstajale le enostavne, statične spletne strani, so danes kompleksne spletne aplikacije nekaj povsem vsakdanjega in vprašanje, kako jih razviti kar se da učinkovito, je vedno bolj v ospredju.

2.1 Zgodovina spleta in spletnega programiranja

Koncept svetovnega spleta se je začel razvijati leta 1980, ko je prve zametke interneta razvil anglež Tim Berners Lee [1]. Internet, kot ga poznamo danes, se zaradi pomankanja interesa ni začel resno razvijati vse do leta 1990. Kljub začetnem pomankanju zanimanja javnosti se je Berners Lee odločil nadaljevati z razvojem treh najpomembnejših komponent interneta in sicer protokola HTTP (Hypertext Transfer Protocol), jezika HTML (HyperText Markup Language) in prvega spletnega brskalnika. Tako se je leta 1991 pojavila prva javna različica jezika HTML, prikazana na prvem spletnem brskalniku. Kmalu po tem so bili razviti tudi drugi brskalniki, med drugim sta leta 1995 v uporabo prišli prvi različici brskalnikov Opera in Internet

Explorer.

Če so bile prve spletne strani le preprosti besedilni dokumenti, so danes to kompleksne spletne aplikacije z veliko funkcionalnostmi. Prvi korak proti zahtevnejšim aplikacijam je bil, poleg že obstoječega jezika HTML, razvoj jezikov JavaScript in CSS (Cascading Style Sheets). Skriptni jezik JavaScript je v uporabo prišel leta 1995 in je tako tlakoval pot dinamičnim spletnim stranem. Istega leta je bila predstavljena tudi prva različica jezika CSS, ki je ponudil več možnosti oblikovanja.

Pravi razvoj interneta, kot ga poznamo danes, se je torej začel leta 1995. Vse od takrat internet ni prenehal rasti. Nastajati so začela družbena omrežja, ki so svoj porast doživela predvsem z ustanovitvijo Facebooka, nastali so zmogljivi spletni iskalniki, kot je Google, v splet pa so se začele povezovati tudi pametne naprave, kot so telefoni in tablice.

Naraščajočim zahtevam uporabnikov svetovnega spleta in vprašanjem o varnosti in zasebnosti želimo zadostiti z vedno novejšimi tehnologijami, izmed katerih so nekatere opisane v nadaljevanju.

2.2 Pregled področja

Pojem spletno programiranje zavzema širok nabor različnih dejavnosti, ki so sestavni del razvoja spletnih rešitev. Razvoj kompleksnejšega spletnega mesta lahko namreč vsebuje vse od grafičnega oblikovanja, razvoja spletnega grafičnega vmesnika, pisanja kode, ki se izvaja na odjemalcu (ang. client-side) ali na strežniku (ang. server-side), pa do nastavitev spletnega strežnika in njegovih varnostnih mehanizmov [2].

Spletni razvoj kot industrija se nenehno razvija. Splet namreč omogoča številne načine uporabe in nudi neskončne možnosti povezovanja uporabnikov med seboj. Za namene razvoja vse večjih spletnih aplikacij so bila razvita najrazličnejša orodja in tehnologije.

Kot že omenjeno je razvoj spletnih aplikacij kompleksen proces, ki zavzema več področij. Glavna delitev pri spletnem programiranju ločuje dve

področji - pisanje kode na strežniku in pisanje kode na odjemalcu. Na področju spletnega programiranja se pojem odjemalec nanaša predvsem na brskalnik, ki stran prikazuje.

2.2.1 Razvoj na strežniški strani

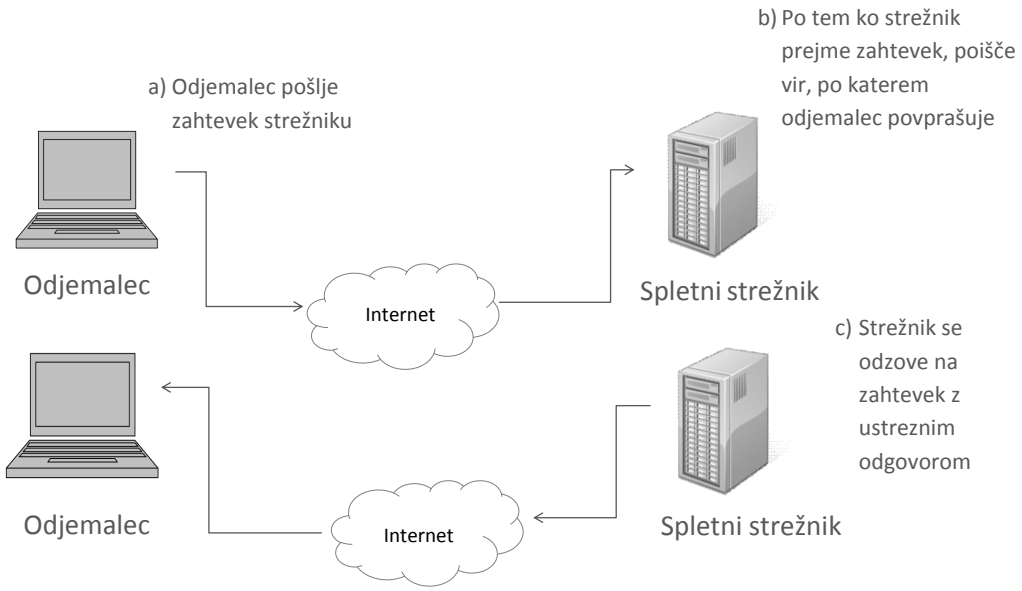
Strežniška koda se shranjuje in izvaja na strežniku, ki gosti spletno aplikacijo ter ni dostopna na odjemalcu oziroma v brskalniku. Bistvena razlika med kodo na strežniku in kodo na odjemalcu je, da se strežniška koda praviloma izvede pred nalaganjem spletne strani oziroma dokumenta HTML, ki spletno stran predstavlja. Skripte, napisane v strežniških jezikih, se torej izvedejo, ko strežnik od odjemalca prejme zahtevo po novih podatkih.

Prve spletne strani so bile statične. Pri statičnih spletnih straneh oziroma aplikacijah strežnik ob prejemu zahtevka le poišče ustrezen vir in ga pošlje brskalniku po standardnih protokolih, kot sta HTTP ali FTP (File Transfer Protocol), pri čemer je uporaba protokola HTTP pogostejša. Ta odgovor nato brskalnik prikaže uporabniku v obliki spletne strani. Taka arhitektura je prikazana na sliki 2.1 [4].

S časom se je spletni razvoj premaknil od statičnih aplikacij k dinamičnim. Bistvo dinamičnih aplikacij je, da se odgovori, ki jih strežnik pošilja brskalniku, generirajo dinamično s pomočjo mnogih strežniških jezikov. Bolj znani in pogosteje uporabljeni jeziki so naslednji [5]:

- Java,
- PHP,
- ASP.NET,
- Ruby on Rails in
- Python.

Strežniška koda torej tako pri statičnih kot tudi pri dinamičnih aplikacijah brskalnik zalaga z dokumenti HTML. Skozi čas se je razmerje med količino kode na strežniku in odjemalcu začelo spreminjati in razvoj se je



Slika 2.1: Arhitektura statične spletne aplikacije

preselil iz povsem strežniških aplikacij proti odjemalskim aplikacijam. Pri tovrstnih aplikacijah vloga strežnika ni več dinamično generiranje dokumentov HTML, temveč strežnik odjemalcu pošilja zgolj podatke. Za pošiljanje le-teh se večinoma uporabljajo spletne storitve, za prikaz podatkov pa poskrbi logika na odjemalcu. Zato se danes strežniški jeziki pogosto uporabljajo zgolj za implementacijo komunikacije s podatkovnimi bazami ter za izvajanje kompleksnejše poslovne logike. S tem se je velik del razvoja preselil na odjemalca, kar je opisano v naslednjem poglavju.

Večino spletnih aplikacij torej sestavljata tako koda na strežniku kot na odjemalcu. Kljub temu, da sta si koda na strani odjemalca in strežnika po nekaterih funckionalnostih sorodni, pa ima vsaka od njiju svoje prednosti. Koda, ki se izvaja na strani odjemalca je primernejša za vse funkcionalnosti, ki zahtevajo veliko interakcije z uporabnikom, medtem ko je strežniška koda večinoma uporabljena za implementacijo poslovne logike oziroma dinamično generiranje podatkov, ki so nato prikazani v spletni aplikaciji [6].

2.2.2 Razvoj na strani odjemalca

Za razliko od strežniške kode se koda na strani odjemalca izvaja lokalno, torej v večini primerov v brskalniku na uporabnikovem računalniku. Koda se v brskalniku tudi shranjuje, medtem ko se strežniška koda shranjuje in izvaja na spletnem strežniku.

Kot že omenjeno je pri sodobnih odjemalskih aplikacijah veliko funkcionalnosti implementiranih na odjemalcu. Za namene razvoja na strani odjemalca je poznanih mnogo različnih tehnologij in orodij. Najpogosteje poteka razvoj v skriptnem jeziku JavaScript, ki skrbi za funkcionalnosti spletne aplikacije. Zelo razširjena je tudi knjižnica JavaScript, imenovana jQuery, ki je bila ustvarjena za enostavnejši in hitrejši razvoj aplikacij. Obstajajo tudi drugi skriptni jeziki, vendar pa so ti večinoma manj razširjeni. Poleg skriptnih jezikov se v vseh primerih na strani odjemalca uporablja tudi kombinacija jezikov HTML in CSS in njunih različic, ki skrbita predvsem za prikaz in izgled spletne strani.

Sodobni brskalniki v skladu s trendi razvoja spletnih aplikacij omogočajo, da se znatna količina kode, ki sestavlja spletno aplikacijo, izvede v brskalniku. Posledica so predvsem uporabniku prijaznejši vmesniki, ki ne zahtevajo nenehnega osveževanja strani ob zahtevi po novih podatkih. Pomembno vlogo pri tej funkcionalnosti ima tehnologija AJAX (Asynchronous JavaScript and XML), ki je pomemben dodatek jeziku JavaScript. AJAX omogoča, da komunikacija med strežnikom in odjemalcem poteka v ozadju in torej ne zahteva osvežitve celotne spletne strani, kar pripomore k boljši uporabniški izkušnji. Kljub napredku in razvoju sodobnih brskalnikov pa podpora skriptnim jezikom še vedno ni enaka v vseh brskalnikih, kar pomeni, da se uporabniška izkušnja med brskalniki lahko razlikuje. Zato je pomembno, da je aplikacija razvita tako, da te razlike niso opazne in ne vplivajo na kakovost uporabniške izkušnje.

Poglavje 3

Sodobne spletne tehnologije

Principi delovanja spletnih aplikacij so skozi čas ostali bolj ali manj nespremenjeni, kljub temu pa se nenehno pojavljajo nove tehnologije in orodja, ki omogočajo hitrejši in bolj strukturiran razvoj zahtevnejših spletnih aplikacij. Poleg novih orodij in tehnologij prihaja do sprememb na področju spletnega programiranja tudi pri arhitekturi spletnih aplikacij, na kar je vplival predvsem hiter razvoj brskalnikov ter porast uporabe pametnih naprav.

Če so včasih brskalniki veljali za lahke odjemalce (ang. thin client), katerih glavna naloga je bila prikazovanje dokumentov, ki so jih preko protokola HTTP pošiljali strežniki, danes brskalniki postajajo vedno bolj zmogljivi. Naprednejše funkcionalnosti brskalnikov se odražajo tudi v vse kompleksnejši arhitekturi spletnih aplikacij. Pomembna komponenta sodobnega spletnega programiranje je tudi prihod pametnih telefonov in tablic. Ker se število uporabnikov, ki do spleta dostopajo s pametnimi telefoni hitro povečuje, se vedno večji pomen pripisuje učinkovitosti spletnih aplikacij, saj je pasovna širina, ki je na voljo za dostop do bogatih spletnih vsebin, vedno bolj omejujoča. Vse to so razlogi, zakaj organizacije, ki se ukvarjajo s standardizacijo in razvojem spletnih tehnologij poskušajo razvijalcem ponuditi sodobnejše tehnologije in orodja. Tako na primer nova generacija jezika HTML, imenovana HTML5, omogoča ogled multimedijskih vsebin v brskalniku brez nameščanja dodatnih vtičnikov (ang. plugin). Posodobitev je doživel tudi jezik CSS, saj

nova različica, imenovana CSS3, ponuja več možnosti oblikovanja. Vsi ti novi standardi in zahteve omogočajo spletnim razvijalcem, da uporabnikom ponudijo bogatejše in odzivnejše aplikacije, ki se sicer izvajajo v brskalnikih, vendar pa se obnašajo kot tradicionalna programska oprema.

3.1 Gradnja sodobnih uporabniških vmesnikov

S porastom raznolikosti naprav, preko katerih dostopamo do spleta, so se pojavile tudi zahteve prilagodljivem oblikovanju, ki omogoča dobro uporabniško izkušnjo na katerikoli napravi. S tem namenom se je pojavilo odzivno spletno oblikovanje (ang. responsive web design). To je pristop k oblikovanju spletnih vsebin, ki omogoča optimalno berljivost in navigacijo po straneh, ne glede na to, iz katere naprave dostopamo [7]. Strani, oblikovane po tem principu, prilagajajo ogrodje strani tako, da uporabljajo na razmerjih osnovane mreže, prilagoljive slike in medijske poizvedbe (ang. media queries), ki jih ponuja jezik CSS. Velikost elementov na strani se tako prilagaja različnim dimenzijam ekranov in ni več določena absolutno, temveč relativno v odstotkih. Za lažji razvoj odzivnih spletnih vmesnikov je bilo razvitih mnogo ogrodij, ki vključujejo in združujejo jezike, kot sta CSS in HTML. Nekatera izmed njih so:

- Twitter Bootstrap,
- Foundation,
- Skeleton in
- YAML.

Twitter Bootstrap in Foundation sta zmogljivejša izmed zgoraj omenjenih ogrodij. Oba vsebujeta sistem odzivnih mrež in nudita močno podporo načrtovanju strani s poudarkom na mobilnih napravah. Bistvena razlika med njima je stopnja oblikovanja, ki ga imajo njuni elementi. Ogrodje Bootstrap

ponuja mnogo oblikovnih predlog in ima že močno oblikovno določene elemente ter je zato boljši za hitro prototipiranje strani. Čeprav je oblikovne lastnosti elementov možno spremeniti, pa ima vseeno večina strani, izdelanih z ogrodjem Bootstrapom, generičen videz. V nasprotju z njim vsebuje ogrodje Foundation veliko manj oblikovanja in je zato primernejši za strani z bolj specifičnim izgledom.

Ogrodji Bootstrap in Foundation sta torej kompleksnejši ogrodji. Podobno ogrodje je tudi YAML, vendar je manj obsežen in ponuja manj funkcionalnosti. Najbolj enostavno ogrodje je Skeleton, ki sicer tudi vsebuje mreže, vendar so le-te fiksnih dimenzij. Tudi ogrodje Skeleton ponuja določeno stopnjo odzivnosti, vendar pa izmed vseh zgoraj omenjenih ogrodij vsebuje najmanj dodatnih elementov in je oblikovno najbolj okrnjen.

Odzivno oblikovanje spletnih vmesnikov ima veliko prednosti tako za uporabnike spletnih aplikacij kot tudi za razvijalce. Glavna prednost je, da ima uporabnik pozitivno uporabniško izkušnjo ne glede na napravo, s katero dostopa do aplikacije. Poleg tega to pomeni, da lahko spletno stran oziroma aplikacijo razvijamo neodvisno od operacijskih sistemov, saj za dostop do take strani potrebujemo le brskalnik. Prihranjeni so torej stroški in čas razvoja, ki bi ga potrebovali za razvoj spletne aplikacije in mobilnih aplikacij za različne operacijske sisteme. Tak pristop pa ne olajša le razvoja, temveč tudi zniža stroške vzdrževanja, saj vzdržujemo le eno spletno mesto hkrati.

Odzivno oblikovanje je torej pomembna komponenta, ki močno poveča kakovost uporabniške izkušnje. Zgoraj našteta ogrodja pa niso edina, ki pripomorejo k sodobnejšim spletnim vmesnikom. Na razvoj in kompleksnost spletnih vmesnikov je pomembno vplival tudi razvoj že dalj časa znanih jezikov, kot sta HTML in CSS, katerih najnovejše različice ponujajo naprednejše gradnike in funkcionalnosti.

3.1.1 HyperText Markup Language

HTML je standardiziran označevalni jezik, ki se ga že od samega začetka uporablja za predstavitev spletnih strani. Tudi danes je, kljub hitremu napredku

svetovnega spleta in mnogim različicam jezika, še vedno glavni gradnik vsake spletne strani.

Osnovni gradniki jezika HTML so elementi, imenovani značke, in njihovi atributi, ki natančneje določajo njihove lastnosti in se največkrat uporabljajo v parih [8]. Brskalniki te značke uporabijo za interpretacijo vsebine spletne strani, ki jo nato prikažejo. Jezik HTML predvsem semantično opisuje strukturo spletne strani in ga zato ne imenujemo programski jezik. Poleg tega omogoča, da v značke vključujemo slike ali druge objekte ter ustvarjamo interaktivne spletne obrazce. Uporaba zgolj osnovnih značk omogoča le izdelavo statične spletne vsebine z osnovnim oblikovanjem. Spletno stran lahko naredimo dinamično z uporabo jezika JavaScript ali njegovih knjižnic, za zahtevnejše oblikovanje spletne strani pa uporabimo jezik CSS.

Jezik HTML se je skozi zgodovino spreminjal in trenutno najnovejša različica je HTML5. Skozi čas je nastala tudi vzporedna veja jezika, imenovana XHTML (Extensible HyperText Markup Language). Ta jezik močneje temelji na jeziku XML (Extensible Markup Language), zato ima strožja sintaktična pravila.

HTML5

HTML5 je najnovejša različica jezika HTML in je predvsem odgovor na dosedanje probleme s standardi, ki so se pojavljali pri uporabi jezikov HTML in XHTML. Razvit je bil torej v želji po standardizaciji enega samega označevalnega jezika, ki bi lahko uporabljal tako sintakso XHTML kot tudi manj strogo HTML. Različica HTML5 je bila razvita tudi z namenom, da dobro deluje na manj zmogljivih napravah, kot so mobilni telefoni in tablice.

HTML5 ponuja mnogo novih elementov, ki nudijo boljšo podporo multimedijskim vsebinam ter vektorskim grafikam in matematičnim formulam [9]. Bistvena prednost uporabe novih elementov je, da za prikaz vsebin niso več potrebni posebni vtičniki. Poleg boljše podpore multimedijskim vsebinam ponuja HTML5 tudi nove značke, ki zamenjujejo pogostej uporabljene elemente in bolje definirajo semantiko spletne strani. Z istim namenom so

uvedli tudi nove atribute oziroma lastnosti značk ter nekatere elemente iz nabora značk odstranili. Poleg izboljšav na področju semantike in podpore multimediji, HTML5 ponuja izboljšave tudi na naslednjih področjih [10]:

- povezljivost,
- "off-line" delovanje in skladiščenje podatkov,
- 2D in 3D grafika in učinki,
- integracija in zmogljivost ter
- povezljivost z napravami.

V nadaljevanju so opisane novosti na vsakem od teh področij.

Povezljivost Različica HTML5 na področju povezljivosti vsebuje nove elemente in funkcionalnosti, kot so Web Sockets, server-sent events in WebRTC. Elementi, imenovani Web Sockets, omogočajo ustvarjanje trajne povezave med odjemalcem in strežnikom, ki služi za prenos podatkov, ki jih aplikacija potrebuje. Funkcionalnost, imenovana server-sent events, omogoča, da strežnik sam pošilja podatke odjemalcu. To pomeni, da se strežnik ne odziva več le na odjemalčeve zahteve, ampak lahko tudi sam začne komunikacijo. Tehnologija WebRTC pa uporabnikom omogoča povezovanje in nadziranje video konferenc neposredno preko brskalnika, kar pomeni, da dodatni vtičniki ali aplikacije niso potrebne.

"Off-line" delovanje in skladiščenje podatkov Nove funkcionalnosti HTML5 vključujejo tudi načine, kako lahko spletna aplikacije deluje brez aktivne internetne povezave. Različica HTML5 zna tudi zaznavati spremembe v aktivnosti povezave ter omogoča shranjevanje strukturiranih podatkov na strani odjemalca, kar lahko pohitri dostope do podatkov. Večina brskalnikov te funkcionalnosti vsaj do neke mere že podpira.

Grafika in posebni učinki Različica HTML5 ponuja nov element, imenovan HTML5 Canvas, ki nudi podporo vektorski grafiki in 3D elementom.

Element canvas omogoča risanje grafik s pisanjem skriptne kode v jeziku JavaScript. Uporablja se ga lahko za izris grafov ali animacij. Ker ta element ni podprt v vseh brskalnikih, še posebej ne v starejših, je dobra praksa, da se v element Canvas vključi tudi alternativen element, ki ga prikažejo brskalniki brez podpore elementu Canvas ali tisti, ki imajo izključen JavaScript.

Integracija in zmogljivost Funkcionalnosti, ki jih ponuja HTML5 na tem področju, vključujejo možnost manipulacije zgodovine brskalnika ter možnost povleci-in-spusti, ki omogoča uporabniku premikanje elementov med spletnimi stranmi. Poleg tega ponuja tudi funkcionalnost, imenovano Web Workers, ki omogoča, da se nekateri deli kode JavaScript izvedejo v ozadju in tako ne upočasnjujejo drugih procesov in dogodkov.

Povezovanje z napravami Najnovejša različica HTML5 nudi boljšo povezljivost med različnimi napravami. Tako HTML5 vsebuje API (Application Programming Interface), ki omogoča uporabo spletne kamere in shranjevanje posnetkov, narejenih z njo. HTML5 vključuje tudi zaznavanje dogodkov, ki se prožijo bo dotiku ekrana, omogoča določanje lokacije uporabnika z uporabo geolociranja ter ponuja zaznavanje orientacije naprave.

3.1.2 Cascading Style Sheets

CSS je standarden jezik, ki se uporablja za oblikovanje dokumentov HTML ali dokumentov XML. Razvit je bil predvsem z namenom ločevanja vsebine dokumenta HTML od izgleda, posledica česar je večja preglednost in strukturiranost. Jezik CSS omogoča tudi, da si lahko več različnih dokumentov deli isto obliko brez nepotrebnega ponavljanja delov kode. Vendar pa jezik CSS ni uporaben le za osnovne možnosti oblikovanja, kot so barve, pisave, razporeditev in velikost elementov. Omogoča tudi prikaz istega dokumenta v različnih oblikovnih oblikah glede na vrsto prikaza ali glede na velikost ekrana.

```
@media screen and (min-width:500px) {
    p {
        font-family: verdana,sans-serif;
        font-size: 14px;
    }
}
```

Izvorna koda 3.1: Primer sintakse medijske poizvedbe

Sintaksa jezika je preprosta in je že od vsega začetka relativno nespremenjena. Principi uporabe jezika CSS torej ostajajo enaki, vendar pa novejša različica ponuja veliko bolj zmogljive načine oblikovanja uporabniških vmesnikov. Najnovejša različica jezika CSS je različica CSS3, ki jo sestavljajo različni tematsko zaokroženi moduli, ki vsebujejo oblikovna pravila različnih generacij. Različica CSS3 ponuja naprednejše načine oblikovanja, kot so:

- napredno senčenje elementov,
- zaokroževanje robov in
- nastavitev slike kot obrobe elementa.

Različica CSS3 vsebuje tudi modul, imenovan CSS Transitions, ki omogoča animiranje prehodov med stanji elementov ter modul, imenovan CSS Animations, ki omogoča animiranje elementov brez sprožanja dogodkov.

Ključne za oblikovanje odzivnih uporabniških vmesnikov so medijske poizvedbe. Ta modul omogoča zaznavanje lastnosti ekrana in prilagajanje izgleda uporabniškega vmesnika temu [12], kar močno izboljša uporabniško izkušnjo in kakovost uporabniškega vmesnika. Medijska poizvedba, katere sintaksa je prikazana v izseku kode 3.1 [13], vsebuje dva atributa, in sicer tip medija (ang. media type), ki označuje vrsto medija, ter značilnost medija (ang. media features), ki določa dodatne lastnosti izbranega medija. Poznanih je več vrst medijev, ki omogočajo na primer oblikovanje dokumenta za navaden tisk ali za tisk v Braillovi pisavi, dokument pa lahko oblikujemo tudi za prikaz vsebine v Braillovi pisavi.

```
@color: #4D926F;

#header {
  color: @color;
}
h2 {
  color: @color;
}
```

Izvorna koda 3.2: Deklaracija spremenljivk v sintaksi LESS

```
#header {
  color: #4D926F;
}
h2 {
  color: #4D926F;
}
```

Izvorna koda 3.3: Primer prevedbe iz LESS v CSS

Leaner CSS

V želji po bolj dinamičnem jeziku CSS je bilo razvitih več jezikov, ki med drugim omogočajo deklaracijo spremenljivk, gnezdenje ter pisanje funkcij in operacij [14]. Prva taka različica je bil jezik, imenovan SASS (Syntactically Awesome Stylesheets), na podlagi katerega je nastal Leaner CSS, ali krajše, LESS. LESS je v svojem bistvu odprtokoden predprocesorski jezik, napisan v jeziku JavaScript. Prevajalnik LESS se lahko izvaja tako v brskalniku kot tudi na strežniku in prevaja skripte [15], napisane v sintaksi LESS v običajno sintakso CSS.

Deklaracija spremenljivk Jezik LESS omogoča deklaracijo spremenljivk, ki so definirane z znakom @ (glej primer 3.2). Med prevajanjem prevajalnik spremenljivke vstavi v generiran dokument CSS, kot je prikazano v primeru 3.3.

```
@the-border: 1px;
@base-color: #111;
@red:        #842210;

#header {
  color: @base-color * 3;
  border-left: @the-border;
  border-right: @the-border * 3;
}
#footer {
  color: @base-color + #003300;
  border-color: desaturate(@red, 10%);
}
```

Izvorna koda 3.4: Funkcije in operacije v sintaksi LESS

Gnezdenje Čeprav jezik CSS omogoča logično gnezdenje elementov, pa ne ponuja gnezdenja same kode. Jezik LESS to omogoča in torej ponuja gnezdenje selektorjev znotraj drugih selektorjev, kar skrajša dolžino kode. Tako kot v vseh primerih, se tudi tu sintaksa LESS prevede v običajen jezik CSS.

Funkcije in operacije Jezik LESS omogoča uporabo funkcij in operacij (glej primer 3.4), ki se nato prevedejo v jezik CSS. Operacije vključujejo deljenje, množenje, seštevanje in odštevanje vrednosti in barv, kar omogoča uporabo kompleksnih relacij med elementi.

Mixins Jezik LESS ponuja tudi funkcionalnost, imenovano mixins, ki omogoča združevanje razredov v en sam razred. Razred deluje kot neke vrste konstanta, lahko pa deluje tudi kot funkcija, saj obstaja možnost podajanja argumentov. Ta funkcionalnost prispeva k večji učinkovitosti kode in manjšemu številu ponavljanj.

Jezik CSS in njegove izpeljanke, kot sta jezika SASS in LESS, so torej osnova za oblikovanje izgleda uporabniških vmesnikov. Z namenom lažjega in hitrejšega razvoja pa so bila razvita ogrodja, ki ponujajo vnaprej definirane razrede, ki poskrbijo za vse pomembne elemente sodobnega uporabniškega vmesnika. Eno izmed takih ogrodij je tudi Twitter Bootstrap, ki je opisan v nadaljevanju.

3.1.3 Twitter Bootstrap

Twitter Bootstrap je brezplačna odprtokodna zbirka elementov, namenjena hitrejši in lažji gradnji spletnih uporabniških vmesnikov. Osnovan je na jezikih HTML in CSS ter ponuja predloge za razporeditev elementov, tipografijo, spletne obrazce, gumbe, navigacijske elemente in še mnoge druge [16]. Ponuja tudi dodatne komponente, ki so implementirane v obliki dodatkov jQuery. Ti dodatki vsebujejo dodatne elemente grafičnih vmesnikov, kot so na primer pogovorna okna.

Pomembna prednost ogrodja Bootstrap je možnost oblikovanja odzivnih grafičnih vmesnikov. To možnost je ponujala že različica 2.0, v različici 3.0 pa je filozofija oblikovanja, ki daje prednost oblikovanju vmesnikov za manjše naprave (ang. mobile first design), še bolj v ospredju. Tak pristop k oblikovanju zagovarja, da naj se uporabniški vmesnik načrtuje najprej za male ekrane in naj se novo vsebino za prikaz na večjih ekranih dodaja naknadno. To je torej bistvena lastnost različice 3.0, ki je trenutno eno najmočnejših ogrodij za izdelavo odzivnih uporabniških vmesnikov [17].

Vključitev Bootstrapa v aplikacijo

Vključitev Bootstrapa v aplikacijo je enostavna (glej primer 3.5 [17]). V kodo vključimo povezave do datotek CSS in JavaScript, ki vsebujejo funkcionalnosti, ki jih ponuja ogrodje Bootstrap. Za uporabo dodatkov JavaScript je potrebno ob razvoju vključiti tudi knjižnico jQuery. Pri vključitvi ogrodja Bootstrap v aplikacijo igra pomembno vlogo značka, imenovana meta. Da je spletna aplikacija primerna za prikaz na različnih vrstah ekranov, je namreč

treba brskalniku določiti, da naj prilagaja velikost spletne strani glede na napravo. Za ta namen se uporabi atribut značke meta, imenovan viewport, s katerim nastavimo začetno širino spletne strani na trenutno širino ekrana.

S tem so osnovne funkcionalnosti in oblikovanje, ki jih ponuja Bootstrap, vključeni v aplikacijo. Oblikovne lastnosti elementov HTML so vsebovane v datoteki bootstrap.css ali njeni enakovredni, a stisnjeni, različici bootstrap.min.css. Bootstrap je kompatibilen z jezikom HTML5, zato lahko uporabljamo nove elemente HTML. Da jim določimo lastnosti, ki jih Bootstrap ponuja, moramo značkam določiti ustrezne razrede CSS, ki so sicer vsebovani v datoteki bootstrap.css.

Ogrodje Bootstrap ponuja privzete oblikovne lastnosti, kot so barve, uporabljena tipografija in oblikovne lastnosti hiperpovezav. Vse te lastnosti je možno na nivoju celotne aplikacije spremeniti v datoteki scaffolding.less. Za to, da je spletna stran samodejno sredinsko poravnana, poskrbi element, imenovan vsebnik (ang. container), katerega širina se dinamično prilagaja širini ekrana. Pomembnejše lastnosti in koncepti, uporabljeni pri gradnji uporabniškega vmesnika z ogrodjem Bootstrap, so predstavljeni v nadaljevanju.

Mreže

Za razporejanje elementov ogrodje Bootstrap uporablja sistem odzivnih mrež, kot je prikazano v izseku kode 3.6. Mreže so bistvena funkcionalnost ogrodja Bootstrap. Za določanje razporeditve mrež glede na velikost ekrana Bootstrap uporablja že prej omenjene medijske poizvedbe. Vsaka vrstica mreže ima lahko največ 12 stolpcev, število prikazanih stolpcev v vrstici pa se povečuje glede na širino ekrana [18]. Ogrodje Bootstrap torej ponuja vnaprej definirane razrede, s katerimi lahko elementom HTML določimo lastnosti odzivnih mrež. V nadaljevanju so našteti osnovni gradniki mrež.

Vsebnik Vsebnik je element, ki poskrbi za sredinsko poravnavo strani. Širina strani je prilagojena širini ekrana. Soroden razred, imenovan container-fluid, pa povzroči, da stran vedno zavzema celotno širino okna.

```
<html>
  <head>
    <title>Bootstrap 101 Template</title>
    <meta name="viewport" content="width=device-width, initial-
    scale=1.0">
    <!-- Bootstrap -->
    <link href="css/bootstrap.min.css" rel="stylesheet" media="
    screen">
  </head>

  <body>
    <h1>Hello, world!</h1>

    <script src="//code.jquery.com/jquery.js"></script>
    <script src="js/bootstrap.min.js"></script>
  </body>
</html>
```

Izvorna koda 3.5: Vključitev funkcionalnosti Bootstrap v aplikacijo

```
<div class="container">
  <div class="row">
      <div class="col-xs-4"> Stolpec 1 </div>
      <div class="col-xs-4"> Stolpec 2 </div>
      <div class="col-xs-4"> Stolpec 3 </div>
    </div>
</div>
```

Izvorna koda 3.6: Primer uporabe sistema mrež

	Zelo majhne naprave **Telefoni (< 768px)**	Majhne naprave **Tablice (≥ 768px)**	Srednje velike naprave **(≥ 992px)**	Velike naprave **(≥ 1200px)**
Orientacija stolpca	Vedno horizontalno	Horizontalno le nad mejno velikostjo ekrana		
Širina stolpca	Auto	60px	78px	95px
CSS razred	col-xs-*	col-sm-*	col-md-*	col-lg-*

Slika 3.1: Lastnosti stolpcev v ogrodju Bootstrap

Vrstica Osnova mreže je vrstica, ki se jo določi z uporabo razreda, imenovanega row. Elementi, ki predstavljajo vrstice, morajo biti obvezno vsebovani v vsebniku.

Stolpec Stolpci so vsebovani v vrsticah in so edini elementi, ki jih vrstica lahko neposredno vsebuje. Ena vrstica lahko vsebuje največ 12 stolpcev. Stolpce določamo z uporabo ustreznih razredov, s katerimi določamo, čez koliko širin stolpcev naj se posamezen stolpec razteza. Za nastanek treh enako širokih stolpcev bi torej uporabili razred .col-xs-4. V tabeli 3.1 [18] je predstavljeno, kako se različni tipi stolpcev obnašajo ob različnih dimenzijah ekranov in katere razrede uporabiti.

Lastnosti posameznega razreda stolpcev se aplicirajo na element, ko širina ekrana preseže eno izmed mejnih vrednosti. Posamezen element ima lahko definiranih več razredov, pri čemer se, če za drugo velikost ekrana ni definiranega razreda, uporabi tisti, ki je definiran. Če je torej na primer na elementu uporabljen razred .col-md-*, se bodo lastnosti tega razreda uporabile pri srednje velikih napravah in tudi pri velikih, v primeru da na istem elementu razred .col-lg-* ni določen.

Ogrodje Bootstrap omogoča tudi določanje odmikov stolpcev. Odmik se določi z razredom CSS po istem principu, kot se določi tip stolpca. S predponami xs, sm, md ali lg določimo, kakšne lastnosti naj ima odmik na različno velikih ekranih, s številko na koncu pa povemo, koliko širin stolpcev naj odmik zavzema. Primer razreda, uporabljenega za odmik, je .col-xs-

offset-4. Poleg odmikov ogrodje Bootstrap omogoča tudi gnezdenje stolpcev in vrstic ter spreminjanje vrstnega reda stolpcev.

Tipografija

Ogrodje Bootstrap vsebuje mnogo razredov za oblikovanje naslovov in drugega besedila. Omogoča na primer oblikovanje sekundarnih naslovov z uporabo razreda .small, uporaba razreda .lead na odstavku pa poskrbi, da je besedilo večje in odebeljeno. Ogrodje Bootstrap ponuja tudi oblikovne predloge za poravnavo besedila, poudarke in okrajšave ter za oblikovanje naslovov pri navajanju kontaktov. Omogoča tudi oblikovanje citatov z navajanjem avtorja, oblikovanje različnih vrst seznamov ter navajanje izsekov programske kode.

Tabele, obrazci, gumbi in slike

Ogrodje Bootstrap ponuja veliko razredov, s katerimi lahko oblikujemo elemente, kot so tabele, vnosna polja in gumbi. Določimo lahko tudi odzivnost slik, katerih velikost se prilagaja velikosti ekrana.

Tabele Osnovno tabelo lahko oblikujemo z uporabo razreda .table, obstaja pa tudi več drugih razredov, s katerimi lahko določimo napredno oblikovanje tabel. Tako lahko na primer z razredom .table-striped naredimo tabelo, kjer so vrstice izmenjujoče poudarjene, z uporabo razreda .table-bordered pa lahko naredimo tabelo z poudarjenimi obrobami.

S posebnimi razredi lahko vplivamo tudi na posamezne vrstice in jih v skladu s kontekstom poudarimo. Tako kot celotno spletno stran lahko tudi posamezno tabelo naredimo odzivno z uporabo razreda .table-responsive.

Obrazci Z uporabo razreda .form-control se nekaterim elementom HTML, kot so vnosna polja ali izbirniki, avtomatsko priredi osnovno oblikovanje. Če te elemente združimo v elementu z razredom .form-group, ogrodje Bootstrap

poskrbi za optimalno razporeditev elementov. Z dodatnimi razredi lahko ustvarimo različno poravnane obrazce in oblikujemo izbirne menuje in polja.

Za namene validacije vnešenih podatkov ponuja Bootstrap naslednje razrede:

- .has-warning,
- .has-success in
- .has-error.

Ti razredi poskrbijo, da vnosna polja dobijo ustrezno barvno obrobo za opozorilo. Poleg barvne obrobe lahko z razredom .form-control-feedback v vnosno polje vključimo tudi grafiko, ki še dodatno ilustrira vrsto napake.

Gumbi Ogrodje Bootstrap omogoča tudi spreminjanje videza gumbov, ki jih lahko z razredi, kot so .btn-succes, .btn-danger ali .btn-alert obarvamo glede na stanje. Gumbom lahko določimo tudi različne velikosti ter jih z razredom .active označimo kot aktivne ali pa jih z uporabo atributa disabled onemogočimo.

Slike Del oblikovanja odzivnih uporabniških vmesnikov je tudi uporaba odzivnih slik, katerih velikost se ustrezno prilagaja velikosti ekrana. Bootstrap to omogoča z uporabo razreda .img-responsive.

Poleg določanja odzivnosti lahko slikam določimo tudi različne oblike. Tako lahko z razredi .img-circle, .img-rounded in .img-thumbnail slike oblikujemo v elipso, krog ali kvadrat.

Twitter Bootstrap je torej močno ogrodje za enostavno gradnjo odzivnih grafičnih vmesnikov. Vendar pa razvoj interneta ni prinesel sprememb le na področju oblikovanja in gradnje spletnih vmesnikov, ampak tudi na področju arhitekture spletnih aplikacij.

3.2 Trendi v arhitekturi spletnih aplikacij

Do sedaj je tradicionalni razvoj spletnih aplikacij vključeval velik del pisanja kode na strežniku. Ta je, po obdelavi podatkov, brskalniku v odgovoru poslal dokument HTML za prikaz. Glede na to, da so bile še nedavno razlike med brskalniki velike, ter da so imeli brskalniki slabo zmogljivo procesiranje jezika JavaScript, je bilo generiranje kode HTML na strežniku dolgo časa smiselno. Vendar pa so brskalniki postali veliko zmogljivejši, kar je omogočilo manjšo količino kode na strežniku in večjo količino procesiranja skriptne kode, ki jo izvaja brskalnik. Porast spletnih aplikacij, ki temeljijo predvsem na kombinaciji uporabe skriptnega jezika JavaScript in jezika HTML, je torej vedno večji, k temu pa pripomore tudi vedno večji delež dostopov do spleta preko pametnih naprav. Vse več funkcionalnosti se torej prenaša na stran odjemalca oziroma v brskalnike.

Ker količina kode JavaScript v sodobnih spletnih aplikacijah narašča, je pri razvoju potrebno veliko pozornosti nameniti strukturi kode, da le-ta omogoča ponovno uporabo posameznih gradnikov. Za lažji razvoj in implementacijo naprednejših funkcionalnosti obstajajo knjižnice JavaScript, ki zmanjšajo količino kode in jo naredijo preglednejšo.

Glavna in najbolj razširjena knjižnica je trenutno jQuery. Ta razvijalcem nudi možnost implementacije različnih funkcionalnosti z veliko manj kode. Poleg tega uporaba knjižnice jQuery odpravlja nekatere nekonsistentnosti med brskalniki, kar močno olajša proces razvoja. Kljub temu, da je knjižnica jQuery zaenkrat še vedno med bolj razširjenimi, pa za različne potrebe obstajajo tudi druge knjižnice. Nekatere od njih so:

- AngularJS,
- AmplifyJS,
- Backbone.js in
- KnockoutJS.

Te knjižnice razvijalcem nudijo dodatne funkcionalnosti, ki jih jQuery ne

```
<!doctype html>
<html ng-app>
    <head>
      <script src="http://code.angularjs.org/angular-1.0.1.min.
    js"></script>
    </head>
    <body>
      <div>
        <input type="text" ng-model="age" />
        <br />
        <h1>You are {{age}} years old!</h1>
      </div>
    </body>
</html>
```

Izvorna koda 3.7: Data binding z AngularJS

podpira. Nekatere izmed teh knjižnic ponujajo podporo arhitekturnemu pristopu, imenovanemu model-pogled-kontroler (ang. model-view-controller), poleg tega pa omogočajo tudi obdelavo in prikaz podatkov na enostavnejši način. Do sedaj so se za prikaz podatkov večinoma uporabljale predloge, kot so na primer elementi GridView, Repeater ali ListView, ki so implementirani s strežniškim jezikom ASP.NET. Taki elementi so razvijalcem olajšali prikaz podatkov, saj te predloge samodejno generirajo kodo HTML, ki podatke v izbrani obliki prikaže. Vendar pa podobne funkcionalnosti omogočajo tudi zgoraj neštete knjižnice, ki bistveno skrajšajo dolžino kode, potrebne za tovrsten prikaz podatkov in razvijalcem na prijaznejši način omogočijo povezovanje podatkov, ki so največkrat v obliki JSON, z gradniki HTML. Kot je prikazano v izseku kode 3.7 [20], lahko z uporabo knjižnice AngularJS tovrstno osnovno funkcionalnost implementiramo z zgolj eno vrstico kode JavaScript. Z uporabo drugih funkcionalnosti pa je možno implementirati tudi kompleksnejše prikazovanje in povezovanje podatkov (ang. data binding) s prikazom.

Pomembno pri razvoju, osredotočenem na odjemalca, je poznavanje br-

skalnikov in njihovih funkcionalnosti, saj se med njimi še vedno pojavljajo nekonsistentnosti. Za optimalno delovanje spletnih aplikacij na vseh brskalnikih je nujno zaznavanje specifičnih funkcionalnosti, ki jih posamezen brskalnik ponuja. Za ta namen obstajo knjižnice, kot je na primer Modernizr, ki zaznava stopnjo podpore za nove elemente jezikov CSS3 in HTML5. Glavno vlogo pri odločitvi za arhitekturo, kjer je večina funkcionalnosti implementirana na odjemalcu, igrata naslednji vprašanji:

- Kdo bo uporabljal spletno aplikacijo in kako dinamična naj bo?
- Ali bo veliko dostopov do spletne aplikacije potekalo iz mobilnih naprav?

Tak način razvoja je torej primeren predvsem za aplikacije, ki želijo uporabnikom ponuditi dinamičnost in odzivnost ter obogateno uporabniško izkušnjo.

3.2.1 Enostranske aplikacije

Enostranske aplikacije (ang. single-page applications) so v svojem bistvu aplikacije, pri katerih je večina funkcionalnosti in logike prestavljena na odjemalca oziroma v brskalnik. Vloga strežnika je torej le še to, da odjemalcu pošilja podatke preko storitev, kot so storitve REST (Representational State Transfer). Ti podatki se večinoma pošiljajo v obliki JSON. Tovrstne aplikacije za delovanje na strani odjemalca potrebujejo zgolj kombinacijo tehnologij HTML, CSS in JavaScript. Enostranske aplikacije izkoriščajo in poudarjajo pomen arhitekture model-pogled-kontroler, ki olajša tako razvoj aplikacije kot tudi testiranje.

Glavna tehnična značilnost tovrstnih spletnih aplikacij je, da se stran nikoli ne osveži v celoti, razen ob začetnem nalaganju. Sodobne enostranske aplikacije imajo nekaj značilnih lastnosti [21], in sicer:

- spletna stran je sestavljena tako, da se posamično nalagajo deli kode HTML in podatki v obliki JSON. Nalaganja celotnega dokumenta ob vsakem zahtevku ni več.

- Koda JavaScript, ki implementira manipulacijo s podatki, logiko delovanja aplikacije in klice AJAX je vsebovana v kontrolerjih, ki v skladu s pristopom model-pogled-kontroler ločujejo pogled od modela.
- Izbira pogledov in navigacija med stranmi poteka brez osveževanja strani in ohranja stanje strani, elementov in podatkov.

Za gradnjo tovrstnih aplikacij je bistvena uporaba pravih tehnologij, ki to omogočajo. Tako so bila razvita ogrodja, ki podpirajo in olajšajo razvoj. Eno najbolj razširjenih ogrodij trenutno je AngularJS, predstavljen v nadaljevanju. Poleg tega pa so za uspešno realizacijo enostranskih aplikacij ključne spletne tehnologije, kot so AJAX, WebSockets in podatkovne strukture, kot je JSON.

WebSocket WebSocket je protokol, ki omogoča polno dvosmerno povezavo preko protokola TCP (Transmission Control Protocol) [22]. Podrobneje je opisan v nadaljevanju v poglavju o Javi EE.

JSON JSON je standardna struktura za hranjenje podatkov. Podatki so shranjeni v obliki parov ključ-vrednost [23] (glej primer 3.8), taka oblika shranjevanja podatkov pa se uporablja predvsem za prenos podatkov med strežnikom in odjemalcem. JSON je tudi alternativa obliki XML shranjevanja podatkov, ki je bistveno kompleksnejša. Čeprav izvorno izhaja iz jezika JavaScript, je danes JSON jezikovno neodvisna oblika shranjevanja podatkov, ki je pregledna in dobro berljiva.

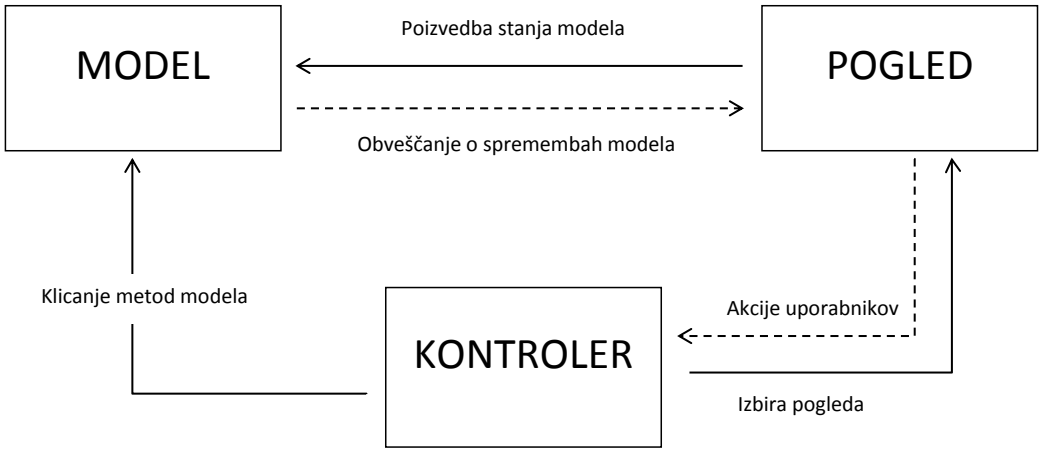
Enostranske aplikacije so večinoma grajene po razvijalskem vzorcu model-pogled-kontroler, ki je predstavljen v nadaljevanju.

3.2.2 Arhitektura model-pogled-kontroler

Model-pogled-kontroler (v nadaljevanju MVC) je arhitekturni razvijalski vzorec, ki se uporablja pri izdelavi spletnih aplikacij [24]. Spletno aplikacijo deli na tri glavne komponente in sicer model, pogled in kontroler, katerih namen

```
{
    "id": 1,
    "name": "Foo",
    "price": 123,
    "stock": {
        "warehouse": 300,
        "retail": 20
    }
}
```

Izvorna koda 3.8: Primer objekta JSON za shranjevanje podatkov



Slika 3.2: Arhitektura model-pogled-kontroler

je ločevanje podatkov od predstavitve le-teh. Medsebojno sodelovanje teh komponent je prikazano na sliki 3.2. Ko kontroler prejme spletni zahtevek, poskrbi za generiranje ustreznega modela. Pogled nato ta model prikaže in sicer največkrat v obliki dokumenta HTML. Kontroler nato zazna morebitne uporabniške akcije in spet poskrbi za ustrezno orkestracijo dogodkov.

Model Model je osrednja komponenta in deluje kot nosilec podatkov. Deluje kot podatkovni vir za pogled in lahko vsebuje tudi poslovno logiko ter funkcije. Model je programsko največkrat predstavljen v obliki programskega razreda ali drugih podatkovnih struktur.

Pogled Pogled je kakršen koli prikaz podatkov uporabniku, ki jih vsebuje model. Naloga pogleda je zgolj prikazovanje podatkov. V spletnih aplikacijah pogled največkrat nastopa kot dokument HTML.

Kontroler Naloga kontrolerja je nadzorovanje toka dogodkov in usklajevanje pogleda z modelom. Pravilna implementacija kontrolerja ne sme vsebovati poslovne logike, ampak je njegova naloga zgolj orkestracija dogodkov. Kontroler tako poskrbi, da se posodobi stanje modela ter poskrbi za ustrezno spremembo pogleda glede na model.

Arhitekturni pristop MVC je bil prvotno sicer razvit za razvoj namiznih aplikacij, vendar je hitro postal priljubljen tudi pri spletnem razvoju. Prva ogrodja MVC so zagovarjala pristop k razvoju, ki je celotno logiko uvrščal na strežnik, pri čemer so brskalniki veljali za lahke odjemalce. Z razvojem tehnologij na strani odjemalca so se pojavila ogrodja, ki omogočajo, da se večino komponent izvede na odjemalcu. Eno takih ogrodij je tudi AngularJS.

3.2.3 AngularJS

AngularJS je odprtokodno ogrodje, razvito pri podjetju Google, ki olajšuje razvoj enostranskih aplikacij, razvitih po principu MVC [25]. AngularJS razširja tradicionalen jezik HTML z namenom, da le-ta bolje služi prikazu dinamičnih vsebin. Za te namene AngularJS uporablja dvosmerno povezovanje podatkov (ang. two-way data binding), ki omogoča avtomatsko sinhronizacijo modelov in pogledov. Ogrodje AngularJS torej v svojem bistvu sledi že prej omenjenemu arhitekturnemu vzorcu MVC in vzpodbuja šibko sklopljenost komponent, kot so podatki, predstavitev podatkov in logika delovanja.

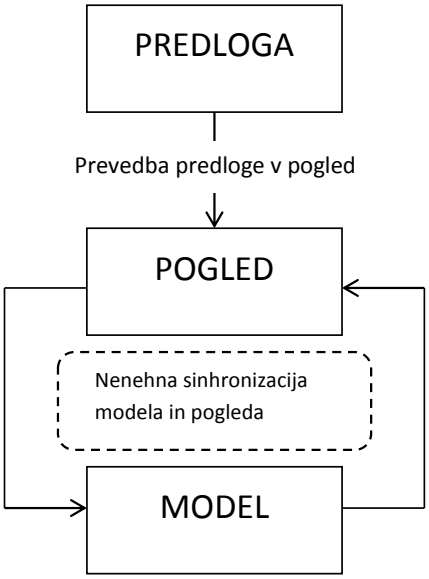
Dvosmerno povezovanje podatkov

Dvosmerno povezovanje podatkov močno skrajša dolžino kode, ki je sicer potrebna za manipulacijo in prikazovanje podatkov. V bistvu gre pri dvosmer-

nem povezovanju podatkov za samodejno sinhronizacijo modela, ki vsebuje podatke, in pogleda, ki podatke prikazuje [26]. Ob spremembi modela se tako samodejno spremeni tudi pogled in obratno, kar pomeni, da je pogled v vseh trenutkih odraz stanja modela.

Pri klasičnem pristopu deluje prikaz podatkov tako, da se predloga, ki je v bistvu dokument HTML, v katerem bodo prikazani podatki, in model združujeta v pogled, ki je šele po združevanju prikazan v brskalniku. Pri tem se spremembe, ki jih uporabnik naredi na pravkar prikazanem pogledu, v modelu ne odražajo samodejno. Da se to zgodi, je pri klasičnem pristopu potrebno pisanje kode, ki poskrbi za sinhronizacijo modela in pogleda. Za razliko od klasičnega pristopa pa AngularJS najprej prikaže predlogo HTML, ki se ob prikazu v brskalniku spremeni v prazen pogled. S tem trenutkom se vzpostavi povezava med pogledom in modelom, ki omogoča nenehno sinhronizacijo pogleda in modela. Dvosmerno povezovanje podatkov je prikazano na sliki 3.3 [26].

Pomemben element pri implementaciji in uporabi dvosmernega povezovanja podatkov je spremenljivka $scope, ki deluje kot neke vrste povezava med kontrolerjem in predlogami HTML. Je objekt, v katerega lahko shranjujemo naše modele s podatki, nato pa se na ta objekt sklicujemo v predlogah. Obseg spremenljivke $scope obsega le okolje znotraj posameznega kontrolerja. Za dostopanje do vsebine te spremenljivke iz drugih kontrolerjev obstaja več načinov, med drugim uporaba globalne spremenljivke $rootScope ali implementacija globalnih funkcij, imenovanih storitve (ang. services). Primer uporabe spremenljivke $scope je prikazan v izseku kode 3.9 [27]. Na istem primeru je razvidno tudi kako definirati AngularJS aplikacijo. Ustvarimo namreč AngularJS modul, ki mu določimo ime (glej vrstico 1). Vsakič, ko v nadaljevanju želimo ustvariti nov AngularJS objekt, kot je kontroler, filter, direktiva ali storitev, za to uporabimo eno izmed obstoječih metod, ki jih AngularJS modul ponuja.



Slika 3.3: Dvosmerno povezovanje podatkov v AngularJS

```
var app = angular.module('app', []);

<!-- Kontroler, kjer definiramo spremenljivko -->
app.controller('MainCtrl', function($scope) {
  $scope.message = 'World';
});

<!-- Predloga za prikaz vsebine spremenljivke -->
<body ng-app="app" ng-controller="MainCtrl">
  Hello, {{ message }}
</body>
```

Izvorna koda 3.9: Uporaba spremenljivke $scope

Direktive

AngularJS vsebuje elemente, imenovane direktive (glej primer 3.10 [27]), ki razširjajo funkcionalnosti osnovnih značk HTML. Z uporabo že obstoječih direktiv lahko razvijalci spreminjajo vedenje elementov HTML, AngularJS pa ponuja tudi možnost pisanja lastnih direktiv. Vse direktive ogrodja AngularJS se začnejo s predpono ng- in jih lahko v elemente HTML vključujemo na več načinov, in sicer kot:

- atribute elementov HTML,
- elemente HTML,
- razrede CSS ali
- komentarje HTML.

Pomembnejše direktive so:

- **ng-app**, ki določi, da naj bo element, ki to direktivo vsebuje, in vsi elementi znotraj njega, del aplikacije AngularJS,
- **ng-bind**, ki povzroči, da se vrednost tega elementa spremeni in postane enaka vrednosti elementa, ki je podan kot atribut direktive,
- **ng-model**, ki omogoča dvosmerno povezovanje podatkov in določa, v katero spremenljivko naj se shrani vrednost tega elementa,
- **ng-class**, ki omogoča dinamično dodeljevanje razredov CSS elementu glede na vrednost izraza v direktivi,
- **ng-controller**, ki določi, kater kontroler naj bo zadolžen za obnašanje tega elementa,
- **ng-repeat**, ki je direktiva, ki ji podamo seznam oziroma zbirko podatkov. Element, ki vsebuje to direktivo se nato generira tolikokrat, kot je podatkov v zbirki,
- **ng-show** in **ng-hide**, ki sta sorodni direktivi, ki prikažeta oziroma skrijeta element, če izraz v direktivi velja, ter
- **ng-switch**, ki glede na pogoje prikaže ustrezno predlogo HTML.

```
<button ng-click="show = !show">Show</button>
<div ng-show="show">
    I am only visible when show is true.
</div>
```

Izvorna koda 3.10: Primer direktive AngularJS

```
<span>Celoten znesek nakupa: {{ 1232.12 | currency }}</span>
```

Izvorna koda 3.11: Primer filtra AngularJS

Filtri

AngularJS ponuja veliko možnosti filtriranja in oblikovanja podatkov, ki jih želimo prikazati. To omogoča funkcija, imenovana filter. Z uporabo te funkcije lahko na primer določimo, da naj bo neka številka prikazana kot denarna valuta. Koda, ki je prikazana v izseku 3.11 [27], na primer povzroči, da se znesek izpiše v obliki $1,232.12. Filtri omogočajo tudi razvrščanje elementov v seznamih glede na določene atribute, ogrodje AngularJS pa omogoča tudi izdelavo lastnih filtrov.

AngularJS storitve

Funkcionalnost, imenovana storitev, je v bistvu globalna funkcija, ki jo lahko z vstavljanjem odvisnosti (ang. dependency injection) vstavimo v katerikoli kontroler in jo uporabimo. Tovrstne funkcije imajo in vzdržujejo le eno samo stanje. Ogrodje AngularJS sicer vsebuje veliko že vgrajenih storitev, možno pa je implementirati tudi lastne, kar je prikazano v izseku kode 3.12. V tem primeru funkciji določimo ime (UserInformation), v njej pa definiramo lokalno spremenljivko name, ki ima vnaprej določeno vrednost. To spremenljivko funkcija vrača. V kateremkoli kontrolerju nato funkcijo uporabimo na način, kot je prikazano na primeru 3.13 [27].

```
app.factory('UserInformation', function() {
  var user = {
    name: "Angular.js"
  };

  return user;
});
```

Izvorna koda 3.12: Izdelava storitve AngularJS

```
app.controller('MainCtrl', function($scope, UserInformation) {
  $scope.user = UserInformation;
});
```

Izvorna koda 3.13: Uporaba storitve AngularJS

Validacija

Funkcionalnosti, ki jih nudi AngularJS, omogočajo tudi hitrejšo in lažjo validacijo uporabniških vnosov na strani odjemalca. Kljub validiranju podatkov na odjemalčevi strani pa je treba poudariti, da je, zaradi relativno enostavne manipulacije kode na strani odjemalca, dobra praksa implementirati validacijo tako na strežniku kot tudi na odjemalcu.

AngularJS nudi lastnosti, ki so vezane na vnosna polja in nam nudijo različne podatke o vnosu v nekem polju. Te lastnosti so naštete in opisane v nadaljevanju, vse pa so povezane z istoimenskimi direktivami in so tipa boolean [28].

- **$valid** nam pove, ali je vnešena vsebina veljavna glede na pravila, ki jih je določil razvijalec.
- **$invalid** nam pove, ali je, glede na pravila, vsebina neveljavna.
- **$pristine** ima vrednost true, če polje še ni bilo uporabljeno.
- **$dirty** ima vrednost true, če je polje že bilo uporabljeno.

Na vseh vnosnih poljih nato nastavimo pravila za validacijo ter z atributom

novalidate izklopimo privzeto validacijo, ki jo opravlja brskalnik. Za validacijo dolžine vnešene vsebine nam AngularJS ponuja direktivi ng-minlength in ng-maxlength, poleg tega pa lahko na vnosnih poljih uporabimo tudi direktivo ng-pattern, ki preveri ustrezno obliko vnosa. Ogrodje AngularJS nato glede na nastavljene direktive za vsako polje nastavi vrednosti zgoraj naštetih lastnosti. V pripadajočem kontrolerju lahko preverimo stanje teh lastnosti in glede na to prikažemo ustrezna opozorila o neveljavnih vnosih.

3.2.4 Razvoj, osredotočen na odjemalca

AngularJS je torej ogrodje, ki je močno osredotočeno na razvoj na strani odjemalca, pri tem pa upošteva MVC razvijalski vzorec. Čeprav se trendi v razvoju spletnih aplikacij vse bolj nagibajo k tovrstnem načinu razvoja, obstaja še vedno veliko vprašanj o učinkovitosti le-tega in o prednostih in slabostih, ki jih tak pristop ponuja.

Zaradi velike raznolikosti naprav in tehnologij, je težnja po šibki sklopljenosti komponent vedno večja. To se močno odraža v arhitekturi aplikacij, kjer so različni nivoji aplikacij čim bolj ločeni med seboj, kar je sorodno tako principu MVC kot tudi ogrodju AngularJS. Arhitektura sodobne spletne aplikacije naj bi torej vsebovala več plasti, kot so na primer:

- shranjevanje podatkov,
- storitve,
- API ter
- predstavitvena plast.

Bistvo ločevanja plasti je, da naj ena plast o tisti nad seboj ve le toliko, kot je res potrebno. Tako lahko na primer predstavitvena plast komunicira s storitvami le skozi API. Predvsem pa je pomembno, da plasti nimajo informacij o plasteh pod njimi. API torej ne sme imeti informacij o tem, kdo bo odjemalec oziroma ne sme imeti nobenega znanja o predstavitveni plasti.

Čeprav je zgornji primer poenostavljen, pa je bistvo predvsem to, da postaja predstavitvena plast vedno bolj ločena od strežnika, ki ne ve ničesar

o odjemalcu. To ločevanje je vedno pomembnejše predvsem zaradi vedno večjega števila različnih odjemalcev, na katerih se predstavitvena plast lahko razlikuje. Ločevanje torej omogoča, da strežnik skrbi le za podatke in poslovno logiko.

Principu šibke sklopljenosti sledi tudi AngularJS, kar je tudi pomembna prednost tega ogrodja. Kljub temu pa ima poleg mnogih prednosti tudi nekaj slabosti, ki pa veljajo tudi za druga ogrodja, ki podpirajo razvoj, osredotočen na odjemalca.

Zaradi kompleksnosti operacij, ki se ob nalaganju strani izvedejo v brskalniku, je nalaganje tovrstne strani počasnejše. Brskalnik mora namreč ob prvem dostopu do aplikacije prenesti vse datoteke, ki jih stran potrebuje za delovanje, vendar je dejstvo, da to ob zmogljivostih današnjih naprav ne predstavlja več večjega problema. Ena izmed težav je tudi optimizacija strani za spletne iskalnike, ki niso sposobni izvajati kode JavaScript, ki jo tovrstne strani vsebujejo in s tem ne prepoznajo vsebine.

Vseeno pa ima razvoj na strani odjemalca bistvene prednosti, zaradi katerih se tak način razvoja vedno bolj uveljavlja. Zaradi prenosa podatkov v obliki JSON in zaradi predpomnenja v brskalnikih se zmanjša količina prenešenih podatkov, kar izboljša tudi zmogljivost strežnika. Ker AngularJS potrebuje zgolj informacije o naslovu spletnih storitev, je predstavitvena plast povsem ločena od strežniške, kar zagotavlja visoko stopnjo neodvisnosti.

Skupaj z jezikoma HTML5 in CSS3 je ogrode AngularJS torej kombinacija, ki omogoča gradnjo spletnih aplikacij, osredotočenih predvsem na odjemalca. Skupaj z odzivnim grafičnim vmesnikom, kar omogoča ogrodje Bootstrap, lahko taka aplikacija povsem nadomesti mobilne aplikacije in s tem tudi poceni in pohitri razvoj. Strežniške tehnologije, ki so opisane v nadaljevanju, so torej le osnova, in skrbijo predvsem za izvajanje poslovne logike in manipulacijo s podatki, ki so izpostavljeni kot spletne storitve.

3.3 Strežniške tehnologije

MVC razvijalski vzorec je princip, ki ga lahko upoštevamo tudi pri razvoju strežniških funkcionalnosti, tudi če se ne odločimo za razvoj, osredotočen na odjemalca. Vendar pa ima razvoj, osredotočen na odjemalca veliko prednosti, posledica tega pa je, da lahko na strežniku pustimo zgolj poslovno logiko in storitve, ki spletno aplikacijo zalagajo s podatki. V nadaljevanju je podrobneje predstavljen jezik Java EE, ki omogoča zmogljivo manipulacijo s podatki v podatkovnih bazah in gradnjo spletnih storitev.

3.3.1 Java EE

Java EE je kratica za Java Enterprise Edition, ki je platforma za razvoj aplikacij in razširitev standardne različice Jave, imenovane Java SE [29]. Java EE se uporablja predvsem za razvoj obsežnejših, večnivojskih poslovno informacijskih aplikacij, ki izražajo izrazito potrebo po zanesljivosti in varnosti [30]. Za namene razvoja takih aplikacij Java EE vsebuje naslednje pomembnejše tehnologije:

- Java Persistence API (JPA),
- Enterprise JavaBeans (EJB),
- Java Message Service (JMS),
- Java Server Faces (JSF),
- Context and Dependency Injection (CDI),
- Java Server Pages (JSP) in Servleti.

V osnovi se Java EE deli na štiri različne vrste komponent, pri čemer za vsako vrsto komponente obstaja vsebnik, v katerem se ta komponenta izvaja. Te komponente so naslednje [30]:

- **apleti** (ang. applet), ki so grafične aplikacije, ki tečejo v brskalniku na odjemalcu,

- **odjemalske aplikacije** (ang. application client), ki so programi, katerih poslovna logika se izvaja na strežniku, grafični vmesnik in aplikacijska logika pa se nahajata na odjemalcu,
- **spletne storitve** (ang. web services), ki se izvajajo na aplikacijskem strežniku in do njih dostopamo preko protokola HTTP,
- **strežniška zrna** (ang. enterprise beans), ki so komponente, ki na strežniku izvajajo poslovno logiko, do njih pa se dostopa preko spletnih storitev SOAP (Simple Object Access Protocol) ali REST.

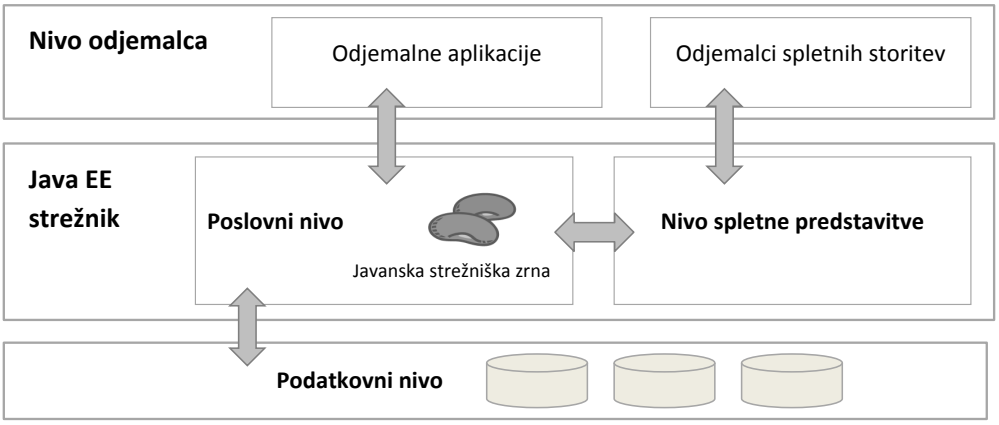
Najnovejša različica Jave EE je Java EE 7, ki na spletnem nivoju ponuja nekaj novosti. Med drugim ponuja podporo za komunikacijo preko protokola WebSocket ter vsebuje standardno knjižnico JSON, ki se uporablja za prikaz in delo s tekstovnimi podatki. V nadaljevanju so podrobneje predstavljene glavne tehnologije Jave EE.

Enterprise JavaBeans

Javanska strežniška zrna (v nadaljevanju EJB) so komponente za modularno izgradnjo poslovno informacijskih aplikacij in so pomemben del Jave EE. Nahajajo se na aplikacijskem strežniku in so del poslovnega nivoja (glej sliko 3.4 [30]). Java EE pozna tri tipe strežniških zrn:

- **sejna zrna** (ang. session beans), ki opravljajo poslovne operacije, orkestirajo transakcije ter upravljajo z dostopi,
- **sporočilna zrna** (ang. message-driven beans), ki so asinhrona zrna in odgovarjajo na zunanje dogodke ter
- **entitetna zrna** (ang. entity beans), ki so imela včasih enako vlogo kot danes entiete JPA (ki so razložene v naslednjem poglavju), vendar so danes zastarela.

Zrna imajo pomembno lastnost in sicer opisno določanje vedenja z uporabo deklarativnih metapodatkov. Metapodatke lahko podamo v obliki XML



Slika 3.4: Umestitev strežniških zrn

```
1 @Stateless
2 public class MojeZrno implements VmesnikZrna {
3   // implementacija poslovnih metod zrna
4 }
```

Izvorna koda 3.14: Anotacija strežniškega zrna

ali z anotacijami, s katerimi prilagajamo vedenje zrn, brez da bi implementirali posebno logiko. Pri določanju lastnosti se uporablja pravilo, da se določi le izjemno vedenje, ki se razlikuje od privzetega (ang. configuration by exception). EJB so šibko sklopljene komponente, ki jih lahko uporabimo večkrat. Ker aplikacije večjih razsežnosti zahtevajo preprosto razširljivost, vsebnik EJB podpira vzdrževanje bazenov (ang. resource pooling), ki omogoča čim večjo ponovno uporabo objektov in virov. V nadaljevanju so podrobneje predstavljena sejna zrna, ki so najpogosteje uporabljena.

Sejna zrna so vrsta strežniških zrn, ki jih uporabljamo za modeliranje posameznih opravil. So v bistvu javanski razredi, ki jim lastnosti zrna določimo z uporabo anotacij @Stateless ali @Stateful, kot je prikazano v izseku kode 3.14. Poznamo dva tipa sejnih zrn in sicer zrna brez stanja (ang. stateless) in s stanjem (ang. stateful). Obstajajo tudi zrna imenovana edinci (ang. singleton), ki označujejo zrno, katerega stanje se vzdržuje od prvega klica do odstranjevanja aplikacije s strežnika, ne glede na uporabnika.

Sejna zrna s stanjem To so zrna, ki tekom interakcije vzdržujejo stanje, ki je običajno povezano s trajanjem ene uporabniške seje. Uporabljamo jih za opravila v več korakih, kjer je interakcija odvisna od stanja v prejšnjih korakih. Tak primer je na primer ohranjanje stanja nakupovalne košarice. Ker zrna ohranjajo stanje, uporabnik vedno dobi isto sejno zrno ob ponovnem klicu metode.

Sejna zrna brez stanja Ta zrna ne vzdržujejo stanja. Uporabljena so zaradi svoje enostavnosti in učinkovitosti in sicer predvsem za generična opravila, ki jih lahko opravimo s klicem ene metode. Čeprav ne ohranjajo stanja povezanega z uporabniško interakcijo, pa lahko ohranjajo stanje, povezano z opravili. Vsebnik EJB vzdržuje bazen sejnih zrn in jih deli uporabnikom, pri čemer ni rečeno, da bo ob vsakem klicu metode uporabnik dobil enako zrno.

Sejna zrna sestavljajo poslovni vmesniki, ki deklarirajo poslovne metode vidne odjemalcem, in so lahko lokalni ali oddaljeni. Privzeto so lokalni, kar lahko spremenimo z uporabo anotacij. Oddaljene vmesnike uporabljamo za dostop do poslovnih metod v drugem izvajalnem okolju, lokalne pa za dostop do istega okolja, v katerem se izvaja aplikacija. Isto zrno ima lahko oba vmesnika hkrati. Druga komponenta sejnih zrn je razred zrna, ki te poslovne metode implementira.

Za izvedbo poslovnih opravil sejna zrna potrebujejo različne vire, ki so lahko druga sejna zrna, podatkovni viri, sporočilne vrste in podobno. Te vire zagotavljamo s postopkom, imenovanim vstavljanje odvisnosti, pri čemer vsebnik vstavi referenco na potrebno komponento. Za vstavljanje virov uporabljamo anotacijo @Resource.

Strežniška zrna omogočajo tudi upravljanje s transakcijami. To so skupine operacij, ki se izvedejo kot enota. Transakcije uporabimo, kadar želimo zagotoviti, da se taka enota izvede celovito. Tako kot vse lastnosti v Javi EE, tudi transakcije določamo deklarativno z uporabo anotacije @TransactionAttribute. Poleg transakcij uporabniška zrna nudijo tudi upravljanje z

varnostnimi nastavitvami. Pri tem z vlogami (ang. security role) določamo dovoljenja za dostop do določenih virov oziroma delov aplikacije.

Java Persistence API

Java Persistence API (v nadaljevanju JPA) je standardni vmesnik za objektno relacijsko preslikavo (ang. object-relational mapping). Je neodvisen, kar zagotavlja prenosljivost med ponudniki ter se enostavno uporablja tako v Javi EE kot tudi v Javi SE [31]. JPA je, kot nam pove beseda API v imenu, v svojem bistvu zgolj specifikacija zahtev, ki jih morajo različne implementacije upoštevati.

Java je objektno orientiran jezik, ki sloni na konceptu razredov. Razredi so med seboj lahko povezani in lahko drug od drugega dedujejo lastnosti. Temu konceptu sledi tudi JPA, saj so glavni elementi entitete JPA, ki so predstavljene kot enostavni razredi in predstavljajo objektno predstavitev tabele v podatkovni bazi. Podatki so v podatkovnih bazah predstavljeni v obliki vrstic in stolpcev v tabelah. Ker je razred lahko v podatkovni bazi predstavljen na več različnih načinov, ročno preslikovanje pa bi zahtevalo veliko truda, je bil razvit JPA.

Namesto relacijskega modela se torej pri razvoju aplikacij uporablja objektni model, kar pomeni, da je vsaka tabela predstavljena v obliki objekta oziroma razreda. Vsaka instanca razreda ustreza eni vrstici v tabeli. Entitete imajo, tako kot tabele v bazi, medsebojne povezave, ki so izražene v obliki metapodatkov. Ti so lahko podani neposredno v razredu v obliki anotacij ali pa v ločeni datoteki XML. JPA torej skrbi za samodejno preslikovanje med objektnim in relacijskim svetom in ponuja trajno shranjevanje podatkov v objektnih strukturah. Za tovrstno shranjevanje podatkov je poznanih več mehanizmov, vendar ima JPA med njimi največ prednosti, ki so, poleg preproste uporabe, naslednje [30]:

- preprosto kreiranje entitet,
- podpora velikim naborom podatkov,

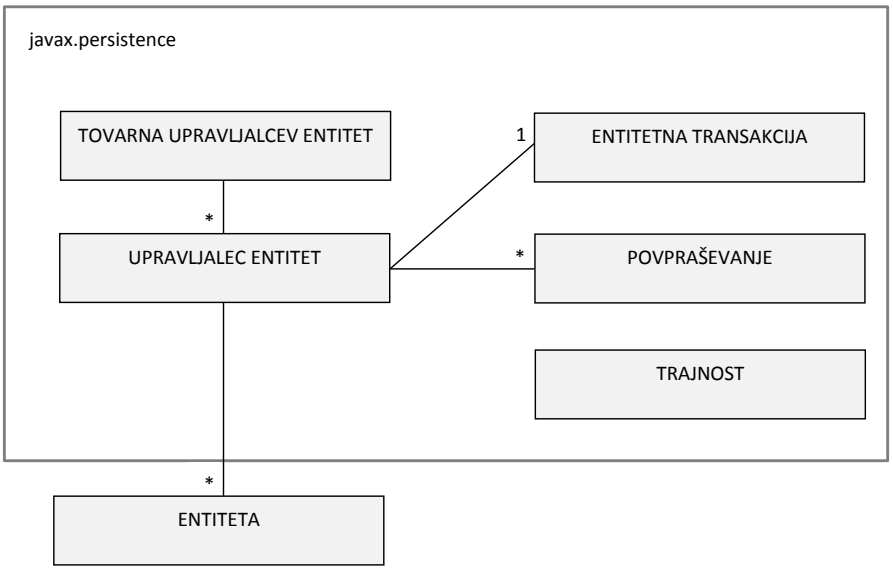
- zagotavljanje konsistentnosti podatkov,
- omogoča uporabo naprednih objektnih konceptov,
- temelji na standardni specifikaciji in se izogiba odvisnosti od ponudnika,
- osredotoča se na relacijske podatkovne baze.

Glavni koncept JPA je že prej omenjena trajnost (ang. persistence). Leta v okviru, ki ga obsega JPA, zajema tri področja [30]:

- samo specifikacijo zahtev, ki specificira izvajanje osnovnih operacij za shranjevanje, posodabljanje, pridobivanje in odstranjevanje objektov iz baze,
- deklarativen način za izvedbo preslikave med objekti in tabelami ter
- povpraševalni jezik JPQL (Java Persistence Query Language) za pridobivanje objektov iz baze, ki ne zahteva pisanja poizvedb SQL (Structured Query Language).

Trajnost je torej posebna lastnost entitet in pomeni, da lahko podatke shranjujemo in pridobivamo s pomočja trajnega medija, kot je na primer podatkovna baza.

Struktura JPA API je definiran v paketu javax.persistence, ki vsebuje različne razrede, ki sestavljajo strukturo JPA (glej sliko 3.5 [30]). Osnovni element je entiteta, ki jo implementira razred Entity in predstavlja zapise v podatkovni bazi. Razred Persistence vsebuje statične pomožne metode za pridobitev instance razreda EntityManagerFactory, ki je tovarna za izdelavo razredov EntityManager. Razred EntityManager je primarni vmesnik JPA, ki ga uporabljajo aplikacije. Vsaka instanca tega razreda upravlja z naborom trajnih objektov in omogoča izvajanje osnovnih (ang. CRUD operations) operacij, kot so kreiranje, pridobivanje, brisanje in posodabljanje objektov v bazah. Razred EntityTransaction omogoča grupiranje operacij nad trajnimi



Slika 3.5: Različne možne predstavitve enega razreda

objekti v enote, ki se lahko v celoti uspešno zaključijo ali pa, v primeru napake, obnovijo začetno stanje podatkovne baze, čemur pravimo razveljavitev transakcije. Del strukture JPA je tudi vmesnik Query. Query API omogoča specifikacijo močnejših iskalnih kriterijev pri uporabi osnovnih operacij. JPA standardizira podporo iskanju tako v jeziku JPQL kot tudi v SQL.

Entitete imajo tudi možnost transakcijskega obnašanja, kar pomeni, da se lahko osnovne operacije izvajajo v okviru transakcijskega konteksta. Poznani sta dve vrsti transakcij, prva so transakcije JTA, kjer vsebnik vsakič začne novo transakcijo, ki se zaključi takoj po koncu izvajanje metode. To se zgodi, če metodo označimo z anotacijo @Requires ali @RequiresNew. Druga vrsta transakcij so "resource-local" transakcije, pri katerih mora programska koda eksplicitno določiti začetek transakcije in jo na koncu potrditi (ang. commit) ali, v primeru napake, razveljaviti spremembe (ang. rollback).

WebSocket

Java EE ponuja tudi API za protokol WebSocket. To je protokol, ki omogoča polno dvosmerno povezavo preko protokola TCP [22]. Pri tradicionalni obliki

komunikacije, ki poteka po principu zahteva/odgovor, odjemalec pošlje zahtevek, strežnik pa mu zagotovi ustrezen odgovor. Komunikacijo vedno začne odjemalec, kar pomeni, da strežnik ne more pošiljati podatkov odjemalcu brez prehodnega zahtevka [32]. Tak način je dobro deloval dokler so odjemalci podajali zahtevke le občasno. Ker so aplikacije vse bolj interaktivne in se vsebina hitreje spreminja je tudi število zahtevkov večje. V takih primerih omogoča protokol WebSocket, skupaj z jezikoma JavaScript in HTML5, boljšo odzivnosti in uporabniško izkušnjo.

Delovanje protokola WebSocket V aplikaciji, ki uporablja protokol WebSocket, strežnik objavi svojo končno točko (ang. endpoint), ki ima svoj URI (Uniform Resource Identifier). Odjemalci se nato na strežnik povežejo tako, da se povežejo na ta URI. Ko je komunikacija vzpostavljena si lahko odjemalec in strežnik simetrično pošiljata sporočila vse dokler je povezava odprta. Odjemalci se večinoma povezujejo na en sam strežnik, medtem ko je na en strežnik lahko povezanih več odjemalcev.

Za razliko od tradicionalnih zahtevkov, ki potekajo preko protokola HTTP, pri protkolu WebSocket preko protokola HTTP poteka le inicializacija komunikacije. Protokol WebSocket ima torej dva dela in sicer rokovanje (ang. handshake) in sam prenos podatkov. Vedno je odjemalec tisti, ki inicializira rokovanje tako, da pošlje zahtevek (glej primer 3.15 [32]) na strežnikovo končno točko. Ta inicializacija poteka preko protokola HTTP, nadaljna komunikacija pa preko protokola TCP. Odgovor strežnika na zahtevek je prikazan na primeru 3.16 [32].

Po uspešnem rokovanju si lahko strežnik in odjemalec med seboj poljubno pošiljata sporočila. Protokol WebSocket torej omogoča, da povezava med strežnikom in odjemalcem ostaja odprta tekom celotne komunikacije. Trenutno je podprt v vseh pomembnejših brskalnikih, podpora protokolu pa je nujna tudi na aplikacijah na strežniku.

Java EE API za WebSocket Java EE vsebuje tudi API za implementacijo protokola WebSocket v spletnih aplikacijah. Strežnikova končna točka je

```
GET /path/to/websocket/endpoint HTTP/1.1
Host: localhost
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: xqBt3ImNzJbYqRINxEFlkg==
Origin: http://localhost
Sec-WebSocket-Version: 13
```

Izvorna koda 3.15: Primer zahtevka za začetek komunikacije, poslan s strani odjemalca

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: K7DJLdLooIwIG/MOpvWFB3y3FE8=
```

Izvorna koda 3.16: Primer strežnikovega odgovora na zahtevek odjemalca

instanca razreda Endpoint, ki je vsebovan v paketu javax.websocket. Ustvarimo jo lahko na dva načina. Lahko razširimo razred Endpoint in prepišemo njegove metode onOpen, onError in onClose, kar je daljši način, ali pa razredu določimo ustrezne lastnosti z uporabo anotacij. V nadaljevanju sta prikazana oba načina ustvarjanja končne točke. V obeh primerih je rezultat končna točka, ki je dostopna na naslovu /echo in se odziva na prejeta sporočila.

Ustvarjanje končne točke z razširjanjem razreda Endpoint je prikazano na primeru 3.17 [32]. V razredu torej implementiramo ustrezne metode in po potrebi kličemo zahtevano poslovno logiko. Ko je razred ustvarjen, ga izpostavimo na nekem naslovu in s tem omogočimo dostop oddaljenim odjemalcem, pri čemer se za vsako povezavo ustvari ena instanca razreda. To omogoča ohranjanje stanj za vsako povezavo.

Drug način ustvarjanja končne točke je uporaba anotacij, kar je krajši in zato tudi preglednejši način. Postopek je prikazan na primeru 3.19 [32]. Z anotacijo @ServerEndpoint določimo URI, na katerem bo točka izposta-

```
1  public class EchoEndpoint extends Endpoint {
2      @Override
3      public void onOpen(final Session session, EndpointConfig
       config) {
4          session.addMessageHandler(new MessageHandler.Whole<String
       >() {
5              @Override
6              public void onMessage(String msg) {
7                  try {
8                      session.getBasicRemote().sendText(msg);
9                  } catch (IOException e) { ... }
10             }
11         });
12     }
13 }
```

Izvorna koda 3.17: Izdelava končne točke z razširjanjem razreda Endpoint

```
1  ServerEndpointConfig.Builder.create(EchoEndpoint.class, "/echo")
       .build();
```

Izvorna koda 3.18: Izdelava končne točke z razširjanjem razreda Endpoint

```
@ServerEndpoint("/echo")
public class EchoEndpoint {
    @OnMessage
    public void onMessage(Session session, String msg) {
        try {
            session.getBasicRemote().sendText(msg);
        } catch (IOException e) { ... }
    }
}
```

Izvorna koda 3.19: Izdelava končne točke z uporabo anotacij

vljena, v tem primeru na naslovu /echo. Pri razširjanju razreda Endpoint določimo ta naslov tako, kot je prikazano na primeru 3.18. Za določanje, katera metode se bo odzivala na kateri dogodek, se uporabljajo naslednje anotacije:

- **@OnOpen**, ki označuje odpiranje povezave,
- **@OnMessage**, ki anotira metodo, ki se odziva na sprejetje sporočila,
- **@OnError**, ki anotira metodo, ki se proži ob napakah ter
- **@OnClose**, ki anotira metodo, ki se proži ob zapiranju povezave.

Končne točke lahko torej pošiljajo in prejemajo sporočila, ki so lahko v obliki besedila ali v binarni obliki. Pri tem igra pomembno vlogo objekt, imenovan Session, ki ga prejmejo metode razreda Endpoint kot parameter. Objekt Session uporabimo, da pridobimo objekt, imenovan RemoteEndpoint. Objekt RemoteEndpoint vsebuje metode, ki jih lahko uporabimo za pošiljanje sporočil v različnih oblikah. Za prejemanje sporočil uporabimo anotacijo @OnMessage, s katero anotiramo metodo, ki se bo na prejetje sporočila odzvala. Posamezna končna točka lahko vsebuje tri metode, ki se bodo odzvale na prejetje sporočila, in sicer eno za vsak tip prejetega sporočila. Kot že omenjeno je prejeto sporočilo lahko v binarni obliki, v obliki besedila, lahko pa je t.i. pong sporočilo.

```
@ServerEndpoint("/chatrooms/{room-name}")
public class ChatEndpoint {
   \\implementacija razreda
}
```

Izvorna koda 3.20: Določanje naslova s parametri

Java API za WebSocket omogoča tudi pretvorbo med javanskimi objekti in sporočili WebSocket. Tako lahko javanski objekt pretvorimo v obliko, ki se lahko pošilja kot sporočilo WebSocket. Take oblike so na primer JSON, XML ali binarna oblika. Omogočen je tudi obraten postopek, kjer se sporočilo pretvori v javanski objekt.

Končna točka je torej dostopna na naslovu, ki ga določimo z anotacijo @ServerEndpoint. API omogoča tudi, da je del naslova podan v obliki parametra, kot je prikazano na primeru 3.20 [32], kjer je parameter določen v zavitih oklepajih. Če je aplikacija na primer dostopna na naslovu http://localhost:8080/chatapp, to pomeni, da se odjemalci lahko povežejo na končno točko z različnimi naslovi, kot so na primer:

- http://localhost:8080/chatapp/chatrooms/currentnews,
- http://localhost:8080/chatapp/chatrooms/music ali
- http://localhost:8080/chatapp/chatrooms/cars.

Metode, ki so anotirane z anotacijami @OnOpen, @OnMessage in @OnClose, lahko te parametre vsebujejo v obliki argumentov. Glede na prejet argument lahko nato metoda izvede ustrezne operacije.

WebSocket je torej protokol, ki omogoča polno dvosmerno komunikacijo, ki poteka preko ene same povezave TCP. Rešuje kar nekaj težav, ki jih ima HTTP [33] in zato tudi REST, ki je v svojem bistvu arhitekturni vzorec za komunikacijo preko protokola HTTP. Kot že omenjeno, mora biti pri komunikaciji HTTP vedno prisoten zahtevek s strani odjemalca, da lahko strežnik pošlje odgovor. Poleg tega komunikacija poteka preko vsakič nove povezave, medtem ko WebSocket omogoča uporabo ene same povezave tekom celotne

komunikacije, kar močno poveča učinkovitost in zmanjša količino prenešenih podatkov. Kljub tem prednostim pa je REST še vedno pogosto uporabljen način gradnje t.i. RESTful aplikacij, predvsem zaradi svoje enostavnosti in funkcionalnosti, ki jih nudi HTTP in jih je pri uporabi protokola WebSocket še vedno treba dodatno definirati.

Java API for RESTful Web Services

Java API for RESTful Web Services ali krajše JAX-RS je komponenta Jave EE za izdelavo RESTful spletnih aplikacij, ki temeljijo na uporabi spletnih storitev. Spletne storitve so tehnologija za integracijo komponent in omogočajo izdelavo šibko sklopljenih komponent [30]. Med seboj si storitve izmenjujejo samo podatke in ne izmenjujejo obnašanja, torej razredov in metod. Komunikacija večinoma poteka po principu zahteva/odgovor, pri čemer se za vsako zahtevo vrnejo zahtevani podatki. Spletne storitve zagotavljajo tehnološko neodvisnost, uporabo standardnih protokolov in dinamično odkrivanje in uporabo storitev. Danes je vse bolj v ospredju uporaba protokola REST, ki je enostavnejša alternativa kompleksnejšim storitvam, kot je protokol SOAP [34].

REST je torej arhitekturni vzorec za branje, brisanje, ustvarjanje ali spreminjanje informacij na strežniku. Značilno zanj je, da komunikacija poteka preko protokola, ki ne vzdržuje stanja, največkrat preko protokola HTTP. Za razliko od protokola SOAP, ki za zahtevke uporablja jezik XML, uporablja REST za dostop do virov naslove URI. Za izvajanje zahtevkov lahko REST uporablja štiri osnovne operacije, in sicer so pri protokolu HTTP to GET, POST, PUT in DELETE. Ker HTTP ne ohranja stanja, je vsak zahtevek popolnoma neodvisen od drugih.

Prednost storitev REST je predvsem enostavnost, omogočajo pa tudi preprosto testiranje. Posamezen vir podatkov lahko pri REST predstavimo v različnih oblikah, kot so JSON, XML, navadno besedilo ali celo slika. Vmesniki REST nimajo standardizacije, zato je uporaba odvisna od dobre dokumentacije. Poleg tega sta avtentikacija in varnost zelo kompleksni, kar

```
1 @Path("/tecaji/{valuta}/")
2 public class ValutniTecajiREST {
3   @GET
4   @Produces("text/plain")
5   public String getTecaj(@PathParam("valuta") String valuta){
6     //implementacija metode
7   }
8 }
```

Izvorna koda 3.21: Primer enostavne storitve REST

pomeni, da je REST manj primeren za prenos občutljivih podatkov.

JAX-RS Storitve REST so v Javi EE implementirane z uporabo orodja JAX-RS. Storitev ustvarimo tako, da ustrezno anotiramo razred storitve z anotacijo @Path, ki določa URI, na katerega bo storitev odgovarjala. V URI lahko po potrebi dodamo tudi parametre. Znotraj razreda z anotacijami @GET, @POST, @PUT, @DELETE označimo metode, ki se prožijo pri ukazih HTTP. Metodam določimo tip parametrov, ki jih sprejemajo in tip odgovora, ki ga vračajo. To storimo z anotacijama @Consumes in @Produces.

Primer preproste storitve REST, zgrajene z uporabo JAX-RS je prikazan v izseku kode 3.21 [30]. V tem primeru bo storitev dosegljiva na naslovu /tecaji, kot parameter pa bo prejela tečaj, po katerem se poizveduje. Parameter je v URI vključen v zavitih oklepajih (glej vrstico 1). Storitev se odziva na zahtevek GET, ob čemer se bo izvedla metoda getTecaj, ki vrača navadno besedilo, kar je določeno z anotacijo @Produces.

REST je torej enostaven način za komunikacijo na spletu, ki omogoča izpostavljanje samo določenih virov. Skupaj z ostalimi predhodno opisanimi Java EE komponentami predstavlja kombinacijo, ki je lahko uporabljena za implementacijo relativno enostavne poslovne logike, sicer pa Java EE ponuja še veliko drugih tehnologij, ki presegajo temo tega diplomskega dela. Tako nam JPA omogoča učinkovito komunikacijo s podatkovno bazo. Enti-

tete JPA so uporabljene v strežniških zrnih, kjer se izvaja poslovna logika, ki manipulira s podatki. Sejna zrna so torej osnova za storitve REST. Razredi storitev dostopajo do metod poslovne logike preko vmesnikov zrn, ki metode izpostavljajo, storitve REST pa nato poskrbijo za komunikacijo med strežnikom in odjemalcem.

V nadaljevanju diplomskega dela je predstavljena izdelava spletne aplikacije, pri čemer so bili uporabljeni do sedaj predstavljeni principi in tehnologije.

Poglavje 4

Razvoj sodobne odzivne spletne aplikacije na primeru

Pomemben del tega diplomskega dela je praktičen prikaz in implementacija spletne aplikacije z uporabo sodobnih principov razvoja. Za ta namen je bil razvit prototip družbenega omrežja, ki uporabnikom omogoča povezovanje z namenom izmenjave časa. Razvit prototip ponuja uporabniku bistvene funkcionalnosti, ki jih mora tovrstno omrežje omogočati, hkrati pa ima družbeno omrežje veliko potenciala za različne razširitve in dodatne funkcionalnosti.

Glavna funkcionalnost razvite aplikacije je izmenjava časa. Omrežje deluje na principu t.i. časovnih bank, pri čemer je glavna valuta čas, oziroma natančneje ura. Uporabniki si tako za vsako uro dela, ki ga opravijo na nalogi drugega uporabnika, prislužijo virtualne točke. Te virtualne točke lahko nato porabijo za objavljanje lastnih nalog, za katere iščejo pomoč oziroma uslugo.

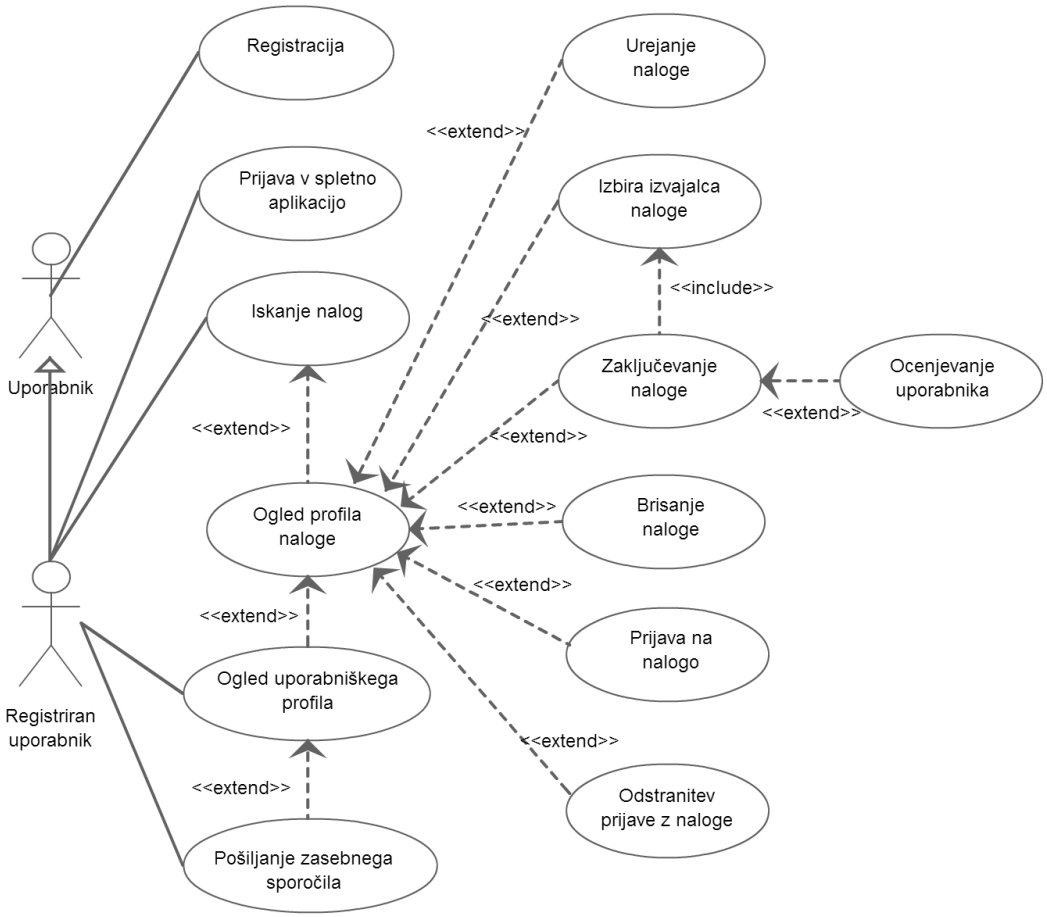
Kot že omenjeno, razvit prototip uporabniku omogoča bistvene funkcionalnosti, ki so potrebne za delovanje tovrstnega družbenega omrežja. Uporabniki se lahko v omrežje registrirajo oziroma prijavijo. S tem se ustvari njihov osebni profil, kjer lahko napišejo kaj več o sebi. Uporabnik ima tudi možnost objaviti novo nalogo, kateri določi opis, lokacijo, kategorijo, informativne začetne in končne datume ter datum, ko se zaključijo prijave na

nalogo. Določi tudi število ur, ki jih naloga zahteva in jo s tem ovrednoti. Poleg objavljanja svojih nalog lahko uporabnik tudi brska po odprtih nalogah, ki so jih objavili drugi uporabniki. Na naloge, ki ga zanimajo, se lahko prijavi. Ko se zapre okno za prijave, lahko lastnik naloge izbere nekoga, ki se mu izmed prijavljenih uporabnikov zdi najbolj primeren. Uporabniki imajo torej tudi možnost pregledovanja profilov drugih uporabnikov in pošiljanja zasebnih sporočil. Ko izbrani uporabnik nalogo konča, lastnik naloge označi nalogo kot zaključeno in oceni ali komentira izbranega uporabnika. Pri tem si izbrani uporabnik tudi prisluži število točk, ki ustreza vrednosti opravljene naloge. Delovanje družbenega omrežja je tudi grafično predstavljeno z diagramom primerov uporabe na sliki 4.1.

4.1 Analiza tehnoloških zahtev aplikacije

Družbena omrežja so spletne aplikacije, ki imajo lahko veliko število uporabnikov in imajo veliko potenciala za dodajanje novih funkcionalnosti. So zelo dinamične aplikacije, ki temeljijo na podatkih, shranjenih v podatkovni bazi. Večinoma torej ne zahtevajo izredno kompleksne poslovne logike, saj le-to večinoma sestavljajo dostopi do podatkovne baze. Spletne aplikacije, na katere uporabniki ne objavljajo občutljivih osebnih podatkov, kot so telefonske številke, številke kreditnih kartic ali bančnih računov, zahtevajo zmerno stopnjo varnosti, ki predvsem ščiti uporabniška gesla in integriteto spletne aplikacije.

Sodobne aplikacije, med njimi tudi družbena omrežja, so aplikacije, do katerih uporabniki pogosto dostopajo iz različnih naprav. Družbeno omrežje mora torej ponujati dobro uporabniško izkušnjo tako na velikih ekranih kot tudi na pametnih napravah, kot so telefoni in tablice, kar zahteva ustrezno načrtovanje grafičnega uporabniškega vmesnika.



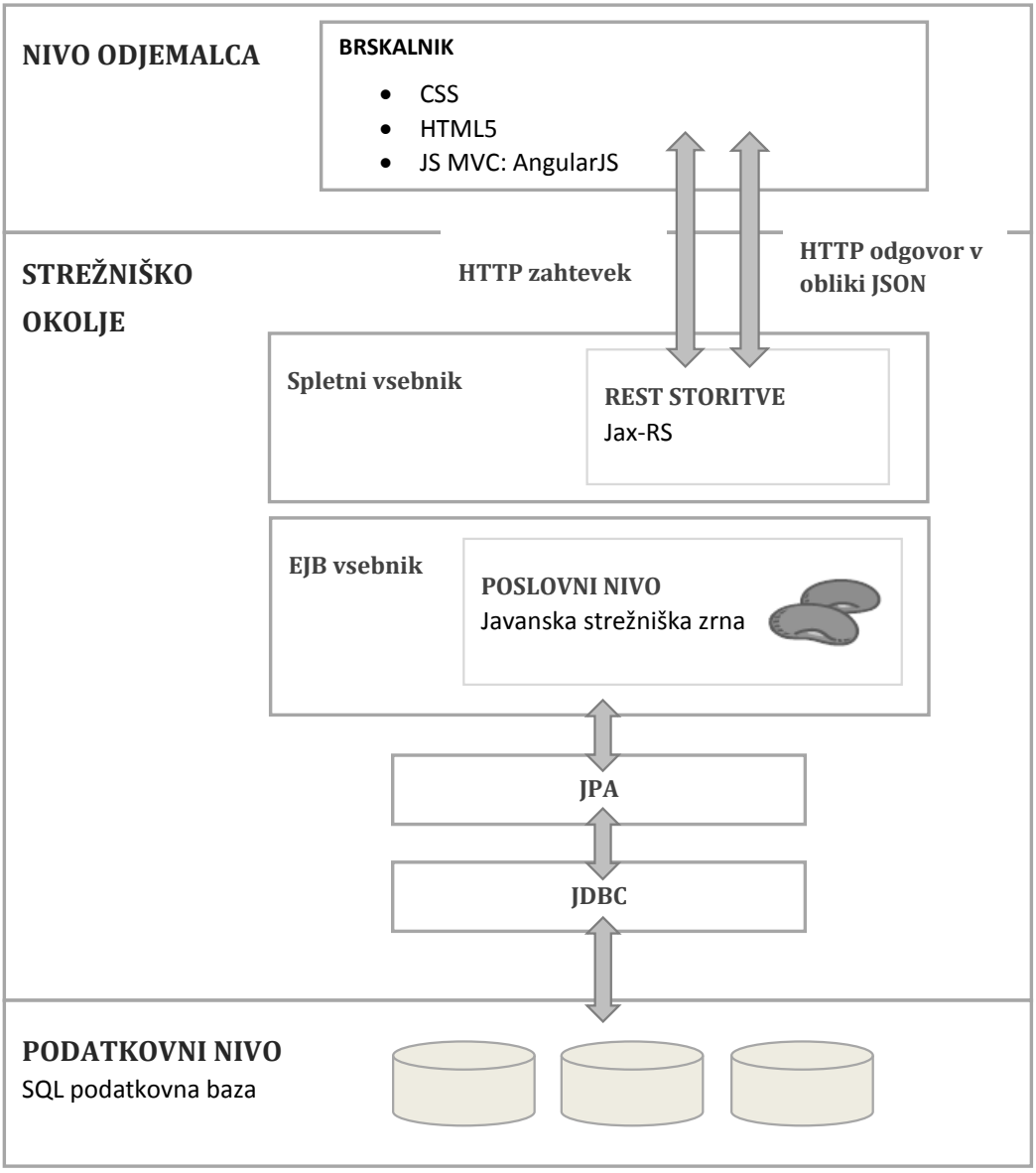
Slika 4.1: Diagram primerov uporabe družbenega omrežja

4.1.1 Izbira tehnologij in arhitektura aplikacije

Predhodno omenjene zahteve so osnova za izbiro tehnologij, uporabljenih pri implementaciji družbenega omrežja. Zaradi želje po odzivnosti in dostopnosti z različnih naprav je uporabniški vmesnik zgrajen z uporabo jezikov HTML5 in CSS ter ogrodjem Bootstrap. Funkcionalnosti spletne aplikacije so implementirane z ogrodjem AngularJS po principu MVC. Ogrodje AngularJS skrbi za orkestracijo dogodkov, prikaz podatkov in komunikacijo s strežnikom ter omogoča visoko stopnjo dinamičnosti aplikacije.

Strežniške funkcionalnosti so implementirane na platformi Java EE. Eden izmed razlogov za izbiro tega jezika je njegova zmogljivost, saj omogoča skalabilnost, kar je pomembno, ker so družbena omrežja aplikacije s potencialno veliko uporabniki in funkcionalnostmi. Poleg tega Java EE ponuja enostavno implementacijo storitev REST, ki so pomemben element, saj omogočajo šibko sklopljenost komponent in razširljivost. Razlog za izbiro je tudi tehnologija JPA, ki predstavlja učinkovit in enostaven način za manipulacijo podatkov v podatkovni bazi.

Ker je aplikacija osredotočena na odjemalca, se na strežniku nahaja zgolj poslovna logika in implementacija spletnih storitev, do katerih dostopa odjemalec. Brskalnik tako ni več lahek odjemalec. Poslovna logika se nahaja v strežniških sejnih zrnih in z uporabo vmesnika JPA manipulira s podatki v podatkovni bazi MySQL. Metode poslovne logike so preko lokalnih vmesnikov zrn izpostavljene razredom storitev REST, zato strežniška zrna predstavljajo osnovo za spletne storitve. Arhitektura aplikacije je grafično prikazana na sliki 4.2.



Slika 4.2: Grafični prikaz arhitekture družbenega omrežja

4.2 Razvoj aplikacije

Izdelava aplikacije se je začela s specifikacijo zahtev in izdelavo diagrama primerov uporabe. Glede na to so bile izbrane tehnologije, katerih izbira je utemeljena v prejšnjem poglavju, zastavljena pa je bila tudi arhitektura družbenega omrežja. Pri razvoju aplikacije so bile uporabljene naslednje komponente in orodja:

- Eclipse, ki je predstavljal glavno razvojno okolje tekom celotnega razvoja,
- aplikacijski strežnik Glassfish za Javo EE ter
- strežnik MySQL za potrebe podatkovne baze.

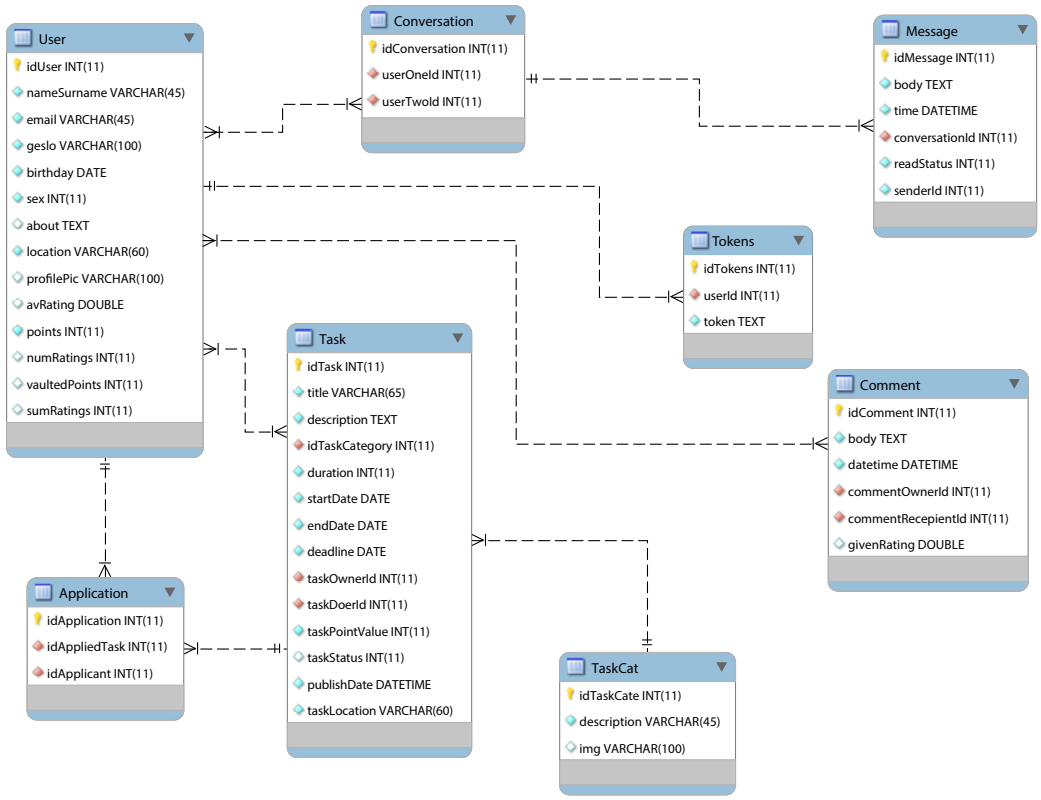
Pri implementaciji družbenega omrežja je bil najprej izdelan skelet (ang. wireframe) spletne aplikacije, ki je služil kot osnova za izgled spletne aplikacije. Na podlagi tega je bil implementiran prototip uporabniškega vmesnika z uporabo jezikov HTML in CSS, ki je že vseboval izbrano oblikovanje in delujoče povezave med podstranmi, vendar je imel še vedno statično vsebino.

4.2.1 Podatkovna baza

Naslednji korak pri implementaciji družbenega omrežja je bila zasnova in postavitev podatkovne baze MySQL. Pred samo izdelavo podatkovne baze je bil zasnovan model podatkovne baze, ki je prikazan na sliki 4.3.

Podatkovna baza vsebuje vse pomembnejše gradnike tega družbenega omrežja, ki ustrezajo zastavljenim funkcionalnostim. Vsebuje torej naslednje tabele:

- Uporabnik (ang. user), ki predstavlja posameznega uporabnika omrežja.
- Pogovor (ang. conversation), ki je povezovalna tabela in predstavlja pogovor med dvema uporabnikoma.
- Sporočilo (ang. message), ki vsebuje zasebna sporočila, ki si jih izmenjujejo uporabniki. Vsako zasebno sporočilo pripada pogovoru med dvema uporabnikoma.



Slika 4.3: Model podatkovne baze, uporabljene pri implementaciji družbenega omrežja

- Žetoni (and. tokens) je tabela, ki vsebuje žetone, ki pripadajo posameznemu uporabniku. Žetoni se uporabljajo pri prijavi uporabnika v sistem in avtentikaciji uporabnika.
- Komentar (ang. comment), ki vsebuje komentarje in pripadajoče ocene uporabnikov.
- Prijava (ang. application) je povezovalna tabela, ki povezuje uporabnika z nalogo, na katero je prijavljen.
- Naloga (ang. task), ki vsebuje vse objavljene naloge na omrežju.
- Kategorija naloge (ang. task category), ki vsebuje kategorije, ki jim naloga lahko pripada.

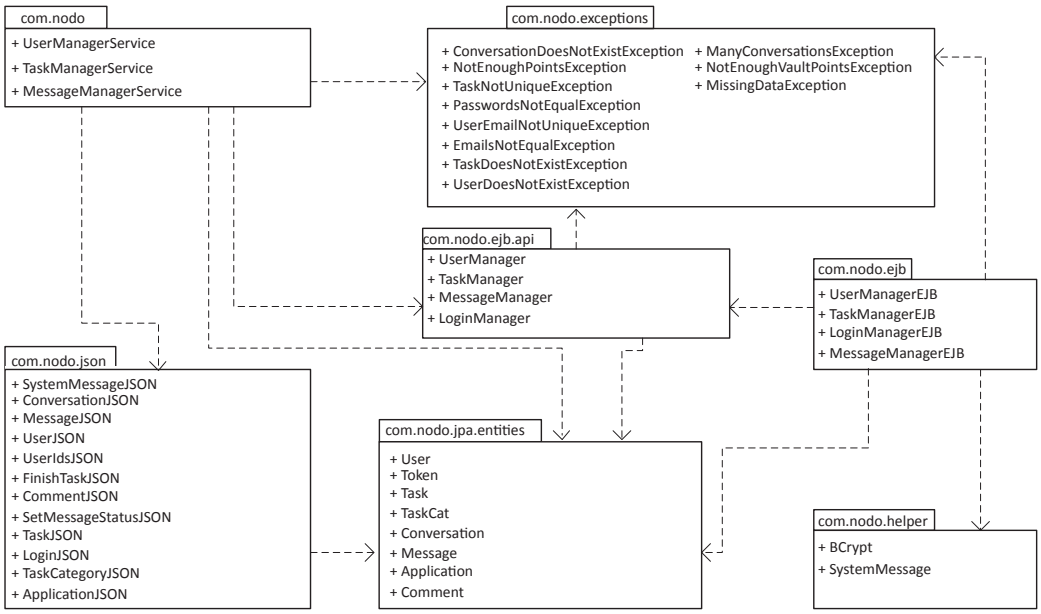
4.2.2 Implementacija strežniških komponent

Postavitvi podatkovne baze je sledila implementacija strežniških komponent, torej poslovne logike in spletnih storitev. Celotna koda na strežniku je organizirana v smiselno zaključene pakete, ki so prikazani na paketnem diagramu 4.4. Vsak od prikazanih paketov vsebuje razrede, ki so zadolženi za posamezne funkcionalnosti aplikacije. V nadaljevanju je razloženo, kaj posamezni paketi vsebujejo in kako so povezani med seboj.

Paket com.nodo.jpa.entities vsebuje entitete JPA, ki ustrezajo podatkovni bazi. Ti razredi imajo mnogo medsebojnih povezav, zato so natančneje predstavljeni z razrednim diagramom na sliki 4.5.

Paket com.nodo.ejb.exceptions je paket, ki vsebuje razrede izjem, ki se prožijo v poslovni logiki. Odziv na prožene izjeme je implementiran v razredih storitev REST.

Paket com.nodo.json vsebuje razrede, ki predstavljajo strukturo sporočil JSON, ki se izmenjujejo med odjemalcem in strežnikom. Uporabljeni so v razredih storitev REST in vsebujejo objekte, ki so definirani v entitetah JPA.



Slika 4.4: Paketni diagram projekta

Paket com.nodo.helper je paket, ki vsebuje pomožne razrede. Ti niso povezani z drugimi razredi, temveč je njihov namen le bolj sturkturirana predstavitev podatkov. Razrede v tem paketu uporablja poslovna logika.

Paket com.nodo.ejb je paket, ki vsebuje razrede sejnih zrn in implementacijo poslovnih metod, ki so definirane v vmesnikih zrn. V teh razredih se prožijo izjeme, dogaja se manipulacija s podatki v bazi, metode pa uporabljajo tudi nekatere pomožne razrede.

Paket com.nodo.ejb.api je paket, ki vsebuje lokalne vmesnike sejnih zrn. Vmesniki definirajo poslovne metode zrn, zaradi česar razredi v paketu potrebujejo razrede izjem in entitete JPA. Vmesniki izpostavljajo poslovne metode, do katerih dostopajo razredi storitev REST.

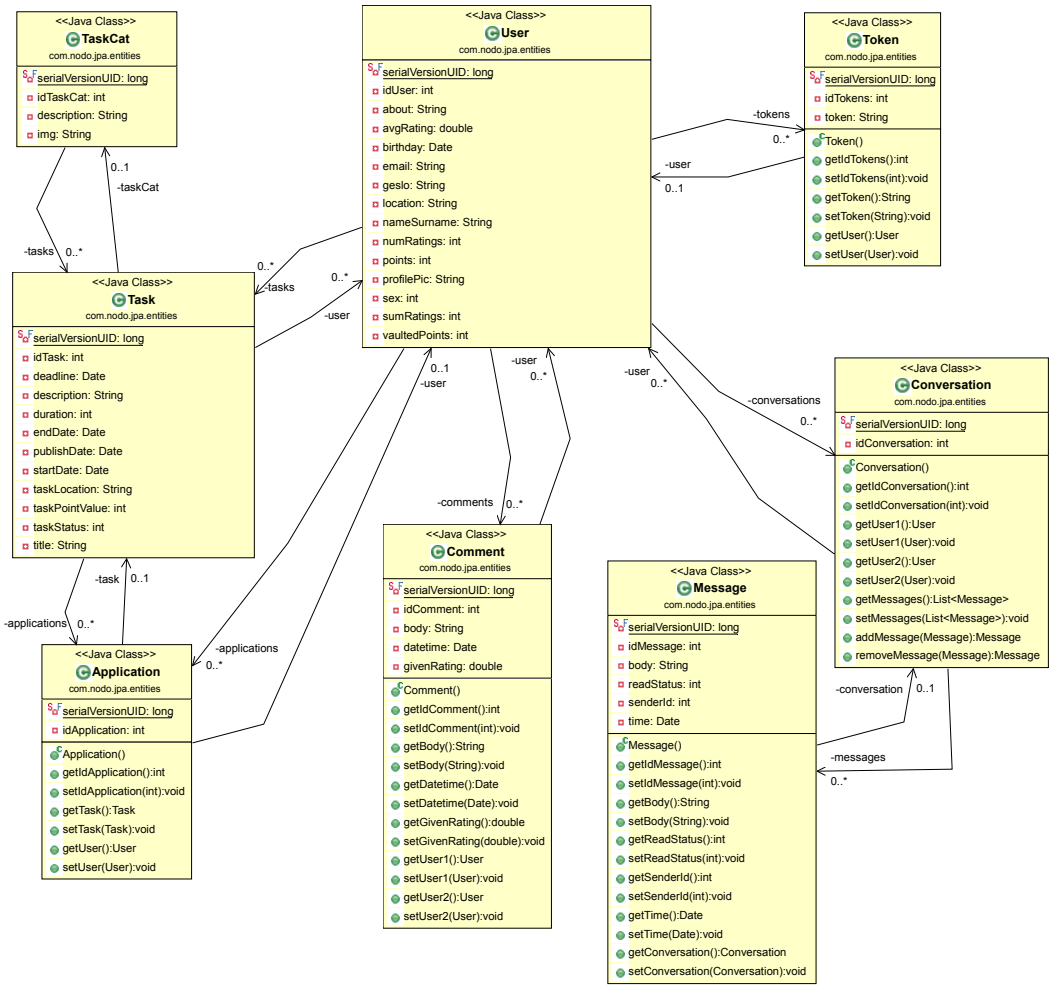
Paket com.nodo vsebuje razrede storitev REST, ki kličejo poslovne metode in izpostavljajo vire odjemalcu. Tu je implementiran tudi odziv na izjeme, ki so prožene v poslovni logiki.

Prvi korak pri implementaciji poslovne logike je bil generiranje entitet JPA. Entitete so bile generirane po principu obratnega inženiringa in s pomočjo orodja Eclipse, generiranju entitet JPA pa je sledila implementacija poslovne logike v sejnih zrnih. Na sliki 4.6 je predstavljen razredni diagram sejnih zrn in njihovih vmesnikov, ki služijo za izpostavljanje implementiranih poslovnih metod. Poslovne metode so implementirane v naslednjih sejnih zrnih:

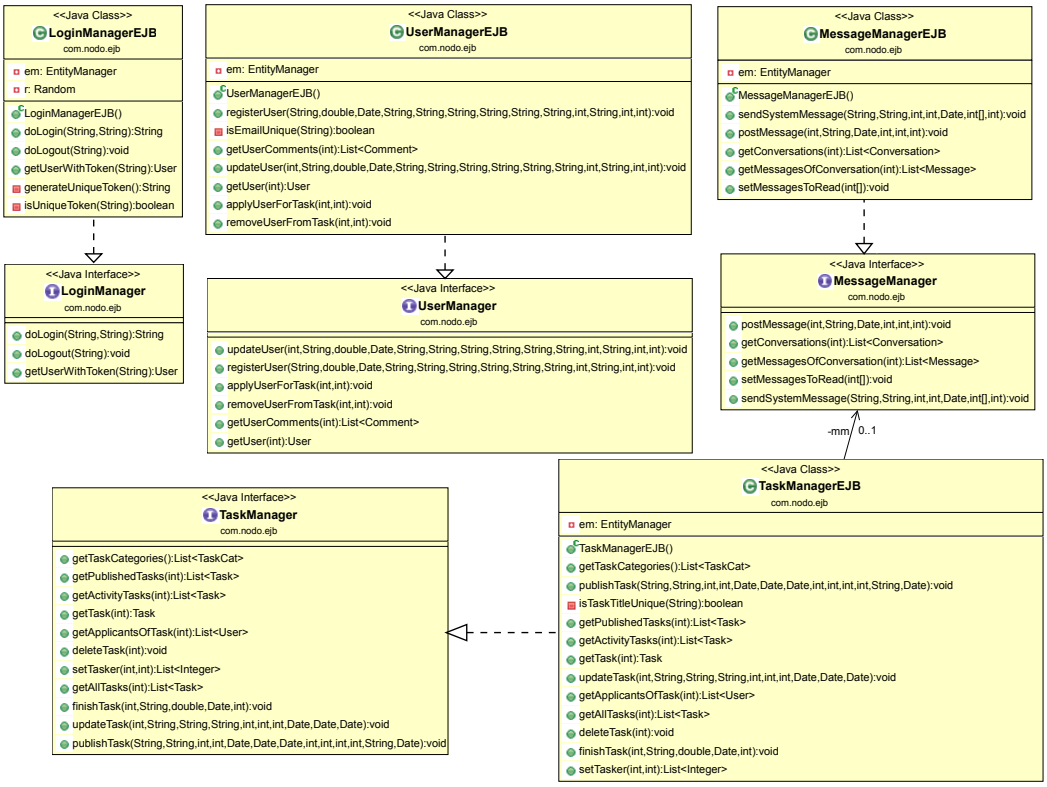
- UserManagerEJB je sejno zrno, ki vsebuje metode oziroma operacije, povezane z uporabniki omrežja.
- MessageManagerEJB je sejno zrno, ki skrbi za upravljanje z zasebnimi sporočili.
- TaskManagerEJB je sejno zrno, ki skrbi za operacije povezane z nalogami.
- LoginManagerEJB je sejno zrno, ki skrbi za registracijo, odjavo in avtentikacijo uporabnikov.

Do izpostavljenih poslovnih metod dostopajo razredi storitev REST, kjer so določeni naslovi URI, na katerih so izpostavljeni viri, do katerih dostopajo odjemalci. Razredi storitev REST nimajo medsebojnih odvisnosti, temveč dostopajo samo do razredov v drugih paketih. Med razvojem spletne aplikacije so bili implementirani naslednji razredi storitev REST:

- UserManagerService je razred, ki vsebuje storitve, povezane z uporabniki.
- MessageManagerService je razred, ki vsebuje storitve za upravljanje z zasebnimi sporočili.
- TaskManagerService je razred, ki vsebuje storitve za upravljanje z nalogami.



Slika 4.5: Razredni diagram entitet JPA



Slika 4.6: Razredni diagram sejnih zrn in njihovih vmesnikov

4.2.3 Implementacija odjemalca

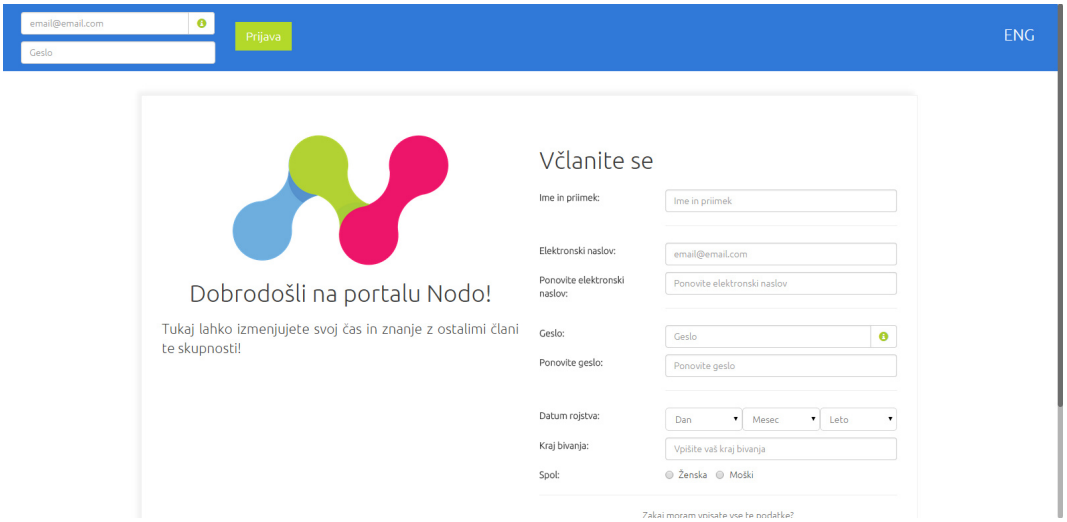
Zadnji korak pri razvoju spletne aplikacije je bil implementacija odjemalskega dela aplikacije. Ker se pri arhitekturnem pristopu, uporabljenem pri razvoju tega družbenega omrežja, večina funkcionalnosti nahaja na odjemalcu, je ta del razvoja temu primerno najbolj obsežen. Polega oblikovanja in implementacije uporabniškega vmesnika obsega tudi implementacijo komunikacije s strežnikom, prikaz prejetih podatkov ter orkestracijo dogodkov.

Razvoj družbenega omrežja je sledil arhitekturnemu pristopu MVC, ki poudarja šibko sklopljenost komponent. Temu primerno je odjemalska aplikacija razdeljena na tri MVC komponente, in sicer model, pogled in kontroler. Kako so vse tri komponente implementirane, je podrobneje opisano v nadaljevanju.

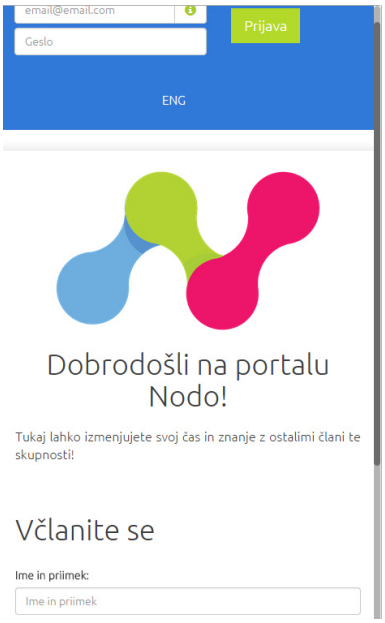
Pogled

Osnova vsake spletne aplikacije je uporabniški vmesnik, ki uporabniku nudi kakovostno uporabniško izkušnjo. Uporabniški vmesnik družbenega omrežja je bil razvit z uporabo jezika HTML5 in ogrodja Twitter Bootstrap, ki olajša odzivno oblikovanje spletnih vmesnikov z uporabo odzivnih mrež. Spletni vmesnik tako ponuja kakovostno uporabniško izkušnjo na različno velikih ekranih. Na slikah 4.7 in 4.8 je prikazan izgled uporabniškega vmesnika za registracijo na velikem ekranu in na pametnem telefonu. Na prvem so elementi razporejeni po širini v dveh stolpcih, na majhnem ekranu pa so vsi elementi zloženi eden nad drugega.

Implementirano družbeno omrežje je v svojem bistvu enostranska aplikacija in temu primerno je organizirana tudi implementacija uporabniškega vmesnika. Le-ta je namreč sestavljen iz enega osrednjega dokumenta HTML, imenovanega index.html, in večih pogledov, ki so prav tako dokumenti HTML. Datoteka index.html določa glavno ogrodje uporabniškega vmesnika in sicer vsebuje zgornji navigacijski meni, vmesni vsebinski del ter nogo strani. Vmesni vsebinski del je del spletne strani, kjer se prikazuje glavna vsebina, ki jo vsebujejo pogledi.



Slika 4.7: Uporabniški vmesnik na velikem ekranu



Slika 4.8: Uporabniški vmesnik na pametnem telefonu

```
<html>
    <head>
        <!-- Glava dokumenta -->
    </head>
    <body ng-app="nodo">
        <div class='row'>
            <div class="navbar navbar-static-top">
              <!-- Zgornji navigacijski meni -->
            </div>
        </div>

        <div class="row mainContainer" ng-view> </div>

        <div id="footer">
          <!-- Noga spletne aplikacije -->
        </div>
    </body>
</html>
```

Izvorna koda 4.1: Določanje mesta za prikaz pogledov

Tak način gradnje spletnih vmesnikov je tipičen za ogrodje AngularJS, ki v ta namen ponuja storitev, imenovano $route. V spletni aplikaciji je to implementirano tako, da je v datoteki index.html uporabljena direktiva, imenovana ng-view (glej izsek kode 4.1), ki poskrbi za vključitev ustreznega pogleda v glavno datoteko. Kateri pogled bo v nekem trenutku prikazan je določeno v datoteki JavaScript, imenovani app.js, kjer je definiran modul spletne aplikacije. Prikaz ustreznega pogleda je implementiran tako, da je za vsak spletni naslov določeno, kateri pogled oziroma datoteka HTML mu pripada. Primer je prikazan v izseku kode 4.2.

Oblikovanje uporabniškega vmesnika, torej glavne strani in pogledov, ter premikanje med pogledi sta bila ključna pri izdelavi pogleda. Kot že omenjeno je glavna naloga pogleda prikazovanje podatkov, ki jih vsebuje model. Naslednji korak je bil torej implementacija modela in kontrolerjev.

```
nodoApp.config(function($routeProvider) {
    $routeProvider
        .when('/Home', {
            templateUrl: 'app/views/Home.html'
        })
        .when('/ChooseWorker/:idTask', {
            templateUrl: 'app/views/ChooseWorker.html'
        })
        .when('/JobProfile/:taskId', {
            templateUrl: 'app/views/JobProfile.html'
        })
        .when('/Profile', {
            templateUrl: 'app/views/Profile.html'
        })
        .when('/UserProfile/:userId', {
            templateUrl: 'app/views/UserProfile.html'
        })
        .when('/JobFinder', {
            templateUrl: 'app/views/JobFinder.html'
        })
        .when('/Messages', {
            templateUrl: 'app/views/Msg.html'
        })
        .when('/JobPublish', {
            templateUrl: 'app/views/JobPublish.html'
        })
        .otherwise({
            redirectTo: 'Home'
        });
});
```

Izvorna koda 4.2: Implementacija izbire ustreznega pogleda

```
var user = {
  "about": "Nekaj o meni",
  "avgRating" : 0,
  "birthday": "1991-03-05",
  "email": "viki@example.com",
  "emailRe": "viki@example.com",
  "from" : "Ljubljana",
  "nameSurname" : "Viki",
  "password": "geslo",
  "passwordRe": "geslo",
  "points": 300,
  "profilePic": "img/slika.jpg",
  "sex" : 0,
  "numberOfRatings": 0,
  "vaultedPoints": 0,
  "sumRatings":0
};
```

Izvorna koda 4.3: Objekt JSON za hranjenje podatkov o uporabniku

Model

Model je pri principu MVC glavni nosilec podatkov, ki so prikazani v pogledu. Na strežniku je to podatkovna baza oziroma razredi JPA, ki ustrezajo tabelam v bazi. Za razliko od Jave EE je AngularJS osnovan na jeziku JavaScript, ki ni objektno orientiran jezik in ne omogoča izdelave razredov. V implementiranem družbenem omrežju so podatki zato predstavljeni v obliki objektov JavaScript. Ti objekti so strukture, ki jim lahko določimo lastnosti in definiramo metode. Eden izmed takih objektov je objekt JSON, ki je namenjen zgolj hranjenju podatkov in ne vsebuje funkcij, temveč le pare ključ-vrednost. Tako je na primer uporabnik v implementirani aplikaciji predstavljen z objektom JSON, ki je prikazan v izseku kode 4.3.

Objekt JSON torej hrani podatke, ti pa so nato ustrezno prikazani v pogledu. Ker pa so podatki, ki jih aplikacija uporablja, shranjeni v bazi na strežniku, mora biti odjemalec sposoben prejemati in pošiljati podatke

```
$http({method: "post", url: "rest/login", data: loginJSON}).
  success(function(data, status, headers) {
      //ob prijavi preusmeritev na uporabnikov profil
      $location.path("/Profile");
    }).
    error(function(data, status, headers) {
      alert('Vnesli ste neustrezne prijavne podatke!');
    });
```

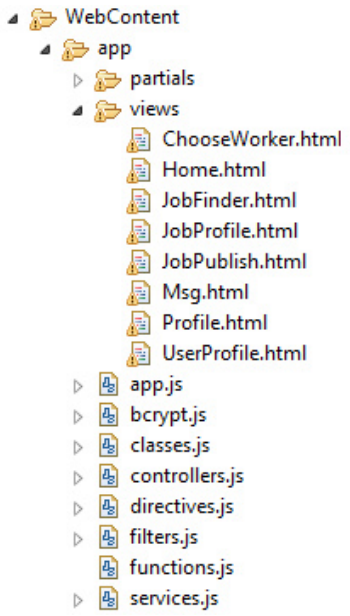
Izvorna koda 4.4: Primer uporabe storitve $http z AngularJS

na strežnik. V implementirani aplikaciji je to realizirano z uporabo storitev REST, zato je to t.i. RESTful aplikacija. Na strežniku so storitve implementirane z uporabo funkcionalnosti Java EE, imenovane JAX-RS, na odjemalcu pa je za te namene uporabljena storitev AngularJS, imenovana $http. To je storitev, ki omogoča asinhrono komunikacijo s strežnikom z uporabo brskalnikovega objekta XMLHttpRequest. Primer pošiljanja podatkov z odjemalca na strežnik je prikazan na primeru 4.4. Tu je prikazan dostop do storitve REST, ki se nahaja na naslovu rest/login. Z zahtevkom POST se proži poslovna metoda za prijavo uporabnika v sistem, pri čemer ji podamo uporabnikove prijavne podatke v obliki JSON.

Model v primeru družbenega omrežja torej predstavljajo objekti JSON, dejanski prenos podatkov pa poteka s pomočjo storitev REST. Vendar pa je vloga modela le hranjenje podatkov, ne pa tudi prikaz in orkestracija dogodkov. To funkcionalnost zajema zadnja komponenta, imenovana kontroler.

Kontroler

Kontroler je bistvena komponenta pri arhitekturi MVC, saj je povezovalni člen med pogledom in modelom. Čeprav ogrodje AngularJS omogoča dvosmerno povezovanje podatkov, ki skrbi za nenehno sinhronizacijo modela, je vseeno potrebno implementirati odzive na dogodke in poskrbeti za njihovo orkestracijo. Vse to se dogaja v kontrolerju, ki je v nekem smislu enakovreden poslovni logiki na strežniku in torej orkestrira dogodke na odjemalcu.

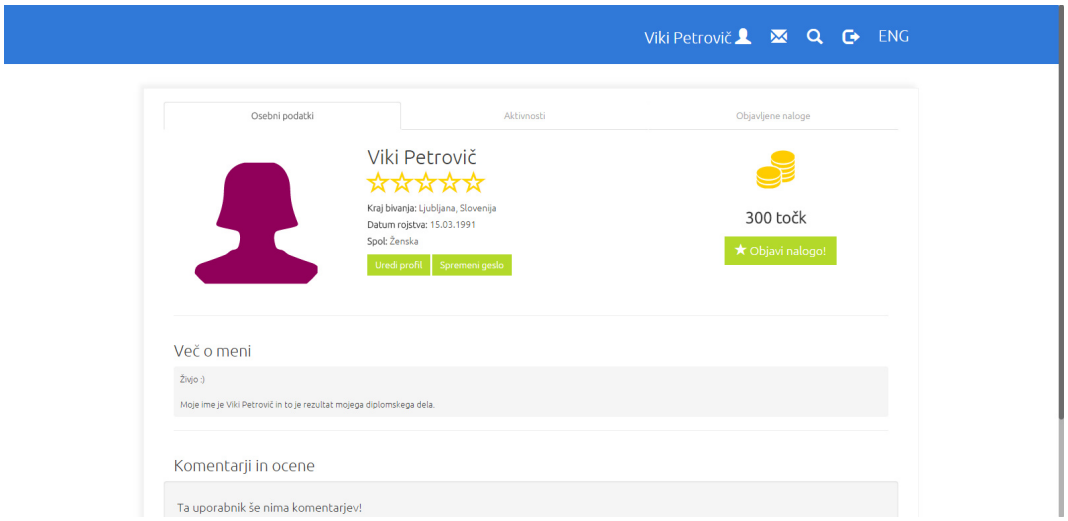


Slika 4.9: Organizacija datotek JavaScript v projektu

Implementacija kontrolerjev je bila bistvena tudi pri implementaciji spletne aplikacije. Bistvo kontrolerjev je, da zgolj orkestrirajo dogodke in s tem sledijo konceptu šibke sklopljenosti. Tako so za manipulacijo pogleda oziroma dokumentov HTML uporabljene direktive, posamezne funkcionalnosti pa so čim bolj modularno implementirane v obliki storitev, ki so enakovredne globalnim funkcijam.

Kontrolerji in vse pripadajoče funkcije so torej čim bolj ločeni med seboj. Temu sledi tudi uporabljena datotečna struktura, ki je prikazana na sliki 4.9. Mapa app je glavna mapa aplikacije in vsebuje datoteke HTML in datoteke JavaScript. Koda vseh kontrolerjev je vsebovana v datoteki controllers.js. Datoteka directives.js vsebuje direktive, datoteka services.js pa storitve, ki skrbijo za dostop do virov na strežniku. Vse ostale globalne funkcije so shranjene v datoteki functions.js. Tudi datotečna struktura torej poudarja ločevanje različnih funkcionalnosti med seboj.

Vsakemu pogledu v implementirani spletni aplikaciji torej pripada eden ali več kontrolerjev, ki skrbijo za delovanje pogleda. V kontrolerjih so definirane funkcije in spremenljivke, ki skrbijo za orkestracijo dogodkov. Bodisi zgolj



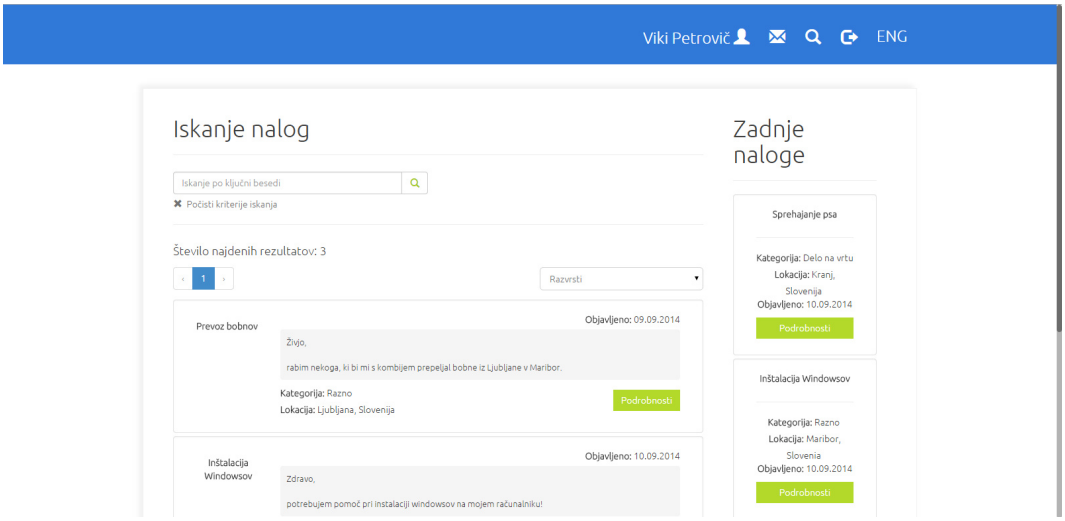
Slika 4.10: Zaslonska slika uporabnikovega profila

določijo vrednosti spremenljivk, ki vplivajo na pogled, ali pa kličejo druge funkcije in storitve.

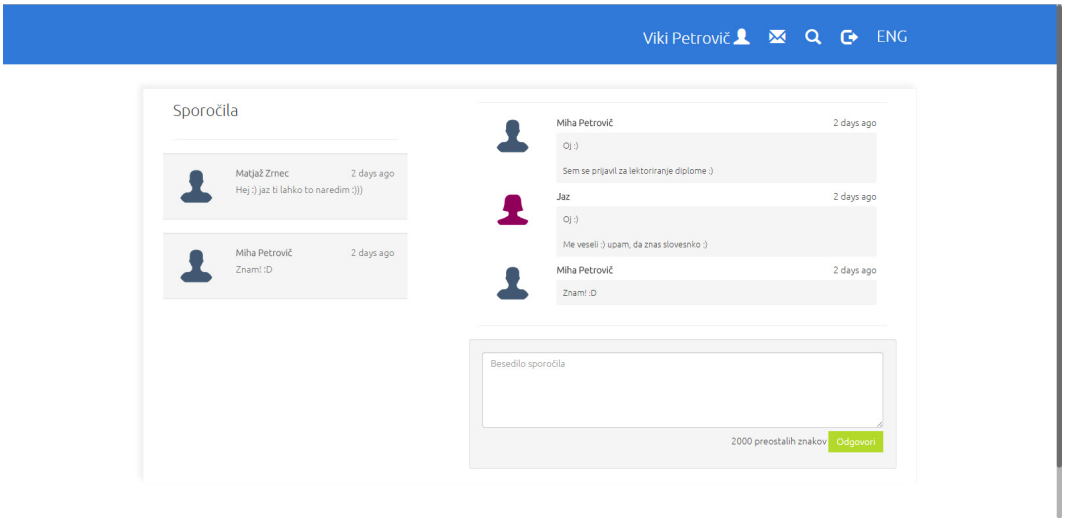
Z implementacijo kontrolerjev, direktiv in storitev na odjemalcu ter implementacijo strežniških komponent je implementirano družbeno omrežje postalo dinamična spletna aplikacija z vsemi zahtevanimi funkcionalnostmi. Spletna aplikacija je predstavljena tudi na slikah 4.10, 4.11 in 4.12, kjer so prikazane zaslonske slike treh bistvenih funkcionalnosti:

- uporabnikovega profila,
- brskanja po nalogah in
- pošiljanja zasebnih sporočil.

Z opisanimi koraki smo torej razvili sodobno odzivno spletno aplikacijo, pri čemer smo uporabili trenutno najbolj aktualne pristope in tehnologije, ki smo jih podrobno predstavili v prejšnjih poglavjih.



Slika 4.11: Zaslonska slika iskanja nalog



Slika 4.12: Zaslonska slika zasebnih sporočil

Poglavje 5

Zaključek

Kot je razvidno iz tega diplomskega dela je razvoj spletnih aplikacij izredno obsežno področje, ki ga je nemogoče v celoti zajeti v enem diplomskem delu. Vsako izmed področij, ki jih razvoj spletne aplikacije obsega, vključuje mnogo različnih pristopov in tehnologij, za namene razvoja pa je na trgu veliko različnih orodij. Odločitev, kakšen arhitekturni pristop in katere tehnologije izbrati, je torej vse prej kot enostavna, vsekakor pa ni dobrih in slabih tehnologij oziroma orodij. Nesmiselno je torej ločevati pravilne pristope od napačnih, pomembno je namreč izbrati tisti pristop, ki najbolj ustreza postavljenim zahtevam.

Tehnologije, predstavljene v tem diplomskem delu niso edine, ki sledijo vzorcu MVC in poudarjajo razvoj, osredotočen na odjemalca, vsekakor pa so trenutno ene najpomembnejših. Kljub temu pa so v tem diplomskem delu predvsem opisani principi, ki niso edinstveni le predstavljenim tehnologijam, temveč v splošnem veljajo za dobro prakso. Tako se tekom diplomskega dela večkrat pokaže, kako pomembno je v resnici ločevanje različnih funkcionalnosti med seboj in zagotavljanje šibke sklopljenosti uporabljenih komponent. Tako arhitekturni vzorec MVC, ki ga uporablja ogrodje AngularJS, kot tudi komponente Java EE, kot so sejna zrna in JPA, ter koncept uporabe storitev REST - vsi v svojem bistvu opravljajo neko zaključeno enoto dela in prevzemajo del odgovornosti za točno določeno funkcionalnost aplikacije.

Da je tak pristop res učinkovit, hitrejši in bolje strukturiran, se je pokazalo tudi pri praktični implementaciji prototipa družbenega omrežja, ki je ob zaključu diplomskega dela delujoča spletna aplikacija, katere pika na i je odziven uporabniški vmesnik, ki je pri sodobnih spletnih aplikacijah že skoraj obvezen.

Iz praktičnega primera je torej razvidno, da ima tovrsten način razvoja aplikacij veliko prednosti, v prid čemur govori tudi razširjenost uporabe. Razvoj spletnih aplikacij, pri katerih se večina funkcionalnosti nahaja na odjemalcu in kjer struktura projekta sledi arhitekturnem vzorcu MVC je torej ustrezen pristop, še posebej če je cilj razviti aplikacijo, ki je zelo dinamična in vsebuje veliko interakcije z uporabnikom. Čeprav to diplomsko delo ponuja pregled nad delom področja razvoja spletnih aplikacij in vključuje praktičen prikaz, pa je to vsekakor področje, ki nudi še veliko možnosti za nadaljnje raziskovanje.

Literatura

[1] A. Lumsden, "A Brief History if the World Wide Web." http://webdesign.tutsplus.com/articles/a-brief-history-of-the-world-wide-web--webdesign-8710, 2012.

[2] Wikipedia, "Web development." http://en.wikipedia.org/wiki/Web_development, 2014. [Online; accessed 20-June-2014].

[3] Wikipedia, "Client-side scripting." http://en.wikipedia.org/wiki/Client-side_scripting, 2014. [Online; accessed 20-June-2014].

[4] Z. Bosnić, *Zbrano gradivo pri predmetu Spletno programiranje*. FRI, 2014.

[5] Wikipedia, "Server-side scripting." http://en.wikipedia.org/wiki/Server-side_scripting, 2014. [Online; accessed 20-June-2014].

[6] "Client Side vs. Server Side." http://www.codeconquest.com/website/client-side-vs-server-side/.

[7] Wikipedia, "Responsive web design." http://en.wikipedia.org/wiki/Responsive_web_design, 2014. [Online; accessed 20-June-2014].

[8] Wikipedia, "Html." http://en.wikipedia.org/wiki/HTML, 2014. [Online; accessed 20-June-2014].

[9] Wikipedia, "Html5." http://en.wikipedia.org/wiki/HTML5, 2014. [Online; accessed 20-June-2014].

[10] "HTML5." https://developer.mozilla.org/en/docs/web/Guide/HTML/HTML5, 2014.

[11] Wikipedia, "Cascading style sheets." http://en.wikipedia.org/wiki/Cascading_Style_Sheets, 2014. [Online; accessed 20-June-2014].

[12] Wikipedia, "Media queries." http://en.wikipedia.org/wiki/Media_queries, 2014. [Online; accessed 20-June-2014].

[13] "CSS Media Types." http://www.w3schools.com/css/css_mediatypes.asp.

[14] Wikipedia, "Less (stylesheet language)." http://en.wikipedia.org/w/index.php?title=LESS_(stylesheet_language)&oldid=613531182, 2014. [Online; accessed 20-June-2014].

[15] "Kaj so SASS, LESS in STYLUS?." http://www.pomagalnik.com/slovar/kaj-so-sass-less-in-stylus/, 2012.

[16] Wikipedia, "Bootstrap (front-end framework)." http://en.wikipedia.org/w/index.php?title=Bootstrap_(front-end_framework)&oldid=612456507, 2014. [Online; accessed 20-June-2014].

[17] S. Fazle Rahman, "Understanding Twitter Bootstrap 3." http://www.sitepoint.com/understanding-twitter-bootstrap-3/, 2013.

[18] "Bootstrap Overview." http://getbootstrap.com/css/.

[19] D. Wahlin, "Explore the New World of JavaScript-Focused Client-Side Web Development." http://devproconnections.com/javascript/explore-new-world-javascript-focused-client-side-web-development, 2012.

[20] D. Wahlin, "Moving from Server-Side to Client-Side Web Development." http://devproconnections.com/javascript/moving-server-side-client-side-web-development, 2012.

[21] Wikipedia, "Single-page application." `http://en.wikipedia.org/wiki/Single-page_application`, 2014. [Online; accessed 20-June-2014].

[22] Wikipedia, "Websocket." `http://en.wikipedia.org/wiki/WebSocket`, 2014. [Online; accessed 20-June-2014].

[23] Wikipedia, "Json." `http://en.wikipedia.org/wiki/JSON`, 2014. [Online; accessed 20-June-2014].

[24] Wikipedia, "Model–view–controller." `http://en.wikipedia.org/wiki/Model-view-controller`, 2014. [Online; accessed 20-June-2014].

[25] Wikipedia, "Angularjs." `http://en.wikipedia.org/wiki/AngularJS`, 2014. [Online; accessed 20-June-2014].

[26] "Guide to AngularJS." `https://docs.angularjs.org/guide`.

[27] J. Rodriguez, "Why does Angular.js Rock?." `http://angular-tips.com/blog/2013/08/why-does-angular-dot-js-rock/`, 2013.

[28] C. Sevilleja, "AngularJS Form Validation." `http://scotch.io/tutorials/javascript/angularjs-form-validation`, 2014.

[29] Wikipedia, "Java platform, enterprise edition." `http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition`, 2014. [Online; accessed 20-June-2014].

[30] M. B. Jurič, *Zbrano gradivo pri predmetu Principi razvoja programske opreme*. FRI, 2014.

[31] Wikipedia, "Java persistence api." `http://en.wikipedia.org/wiki/Java_Persistence_API`, 2014. [Online; accessed 20-June-2014].

[32] E. Jendrock, R. Cervera-Navarro, I. Evans, K. Haase, and W. Markito, "The Java EE 7 Tutorial." `http://docs.oracle.com/javaee/7/tutorial/doc/home.htm`, 2014.

[33] A. Gupta, "REST vs WebSocket Comparison and Benchmarks." http://blog.arungupta.me/2014/02/rest-vs-websocket-comparison-benchmarks/, 2014.

[34] J. Mueller, "Understanding SOAP and REST Basics." http://blog.smartbear.com/apis/understanding-soap-and-rest-basics/, 2013.